SELF-ADAPTIVE FEDERATED LEARNING IN INTERNET OF THINGS SYSTEMS: A REVIEW

Abdulaziz Aljohani
School of Computer Science and Informatics
Cardiff University, UK
`aljohania1@cardiff.ac.uk`

Omer Rana
School of Computer Science and Informatics
Cardiff University, UK
`ranaof@cardiff.ac.uk`

Charith Perera
School of Computer Science and Informatics
Cardiff University, UK
`PereraC@cardiff.ac.uk`

March 22, 2024

ABSTRACT

In recent years, federated learning (FL) and the Internet of Things (IoT) have enabled many artificial intelligence (AI) applications. The new paradigm brings many benefits over conventional machine learning (ML) and deep learning (DL) because it moves model training to the edge. Even though FL proves its feasibility, the constant change in the environment of IoT systems makes it difficult for FL to converge quickly and work as promised. Consequently, a self-adaptive approach is required to automatically react to context changes and maintain system performance. We provide a systematic overview of current efforts to use self-adaptation approaches for FL in the IoT context. In particular, we review different computing disciplines such as self-adaptive systems(SAS), feedback controls, IoT, and FL. Additionally, we present (i) a multidimensional taxonomy to highlight the main characteristics of a self-adaptive FL system and (ii) a conceptual architecture for self-adaptive FL in the IoT context and apply it to anomaly detection (AD) for smart homes. We conclude by describing the motivations, implementations, applications, and challenges of the self-adaptive FL system in the IoT context.

***K*eywords** Federated Learning, Internet of Things, Self-adaptation

1 Introduction

The Internet of Things (IoT) facilitates the rapid advancement of smart applications that transform cities, homes, and workplaces into interconnected, efficient, and responsive spaces [1]. Statistics indicate that the total number of IoT devices is expected to reach 125 billion by 2030. [2]. The significant expansion of the IoT network increases the number of artificial intelligence (AI) applications, which heavily rely on the massive amount of data that were unavailable before. [3]. However, because of the importance of data privacy, most of these data are private for AI applications. Therefore, Google introduced the concept of federated learning (FL) to overcome these obstacles, enabling AI applications to use these data without violating data privacy. The adaptation of FL has expanded to many areas. For instance, several medical organizations may collaborate to develop machine learning (ML) diagnostic models without sharing patients' health records. Smart industries can also benefit by actively sharing model training with similar manufacturers to enhance early fault tolerance in the production line. Domestic users can also benefit from using FL to prevent cyber-physical attacks on domestic IoT devices. To provide a proper notation for this new paradigm, the term "*Internet of Federated Things (IoFT)*" was introduced to explain the relationship between IoT and FL and the explosive interest that has been generated during the last few years [4]. IoFT allows devices to work together to extract knowledge and create intelligent analytics and models while storing their data locally. In addition to lowering privacy concerns, this paradigm shift introduces several inherent benefits, such as cost-effectiveness, diversity, and less computation. However, because of the unpredictable nature of operating environments, such as changing weather and uncertainty around service delivery, IoFT has been displaying increasing complexity. Self-adaptation evolved to allow systems to respond autonomously to shifting environments and preserve the necessary level of service quality [5]. In this paper, we demonstrate the efforts of the scientific community to overcome IoFT challenges using the self-adaptation property. To enhance the reading of this paper, we have included all the abbreviations used in Table 1.

1.1 Existing Surveys

Numerous studies have demonstrated the viability of self-adaptive systems (SAS) in various contexts. We provide a summary of previous research on SAS. Table 2 shows the analysis of self-adaptive IoFT in the literature. The effort to review SAS and its properties has motivated many researchers, especially after IBM proposed the vision of autonomic computing in 2001 [5]. The proposed manifesto demonstrated the urgent need to implement a system that can operate autonomously without human intervention to overcome the complexity of the system. Consequently, their continuous operation became difficult for humans to manage. Early work has been conducted to review autonomic computing and its application [6]. The authors discuss various aspects of SAS, including its motivations, methodologies, and applications. However, their paper should have included a section on SAS evaluation. Moreover, they present the main properties and definitions of self-management systems. Although their study partially discusses the use of a self-management concept in distributed systems, FL should be mentioned. In [7], Salehie and Tahvildari provide an extensive summary of self-adaptive software research by identifying related fields and notable research initiatives. They used a question-based approach (when, what, how, and where questions) to identify the main properties of SAS. Krupitzer et al. [8] outlined further analysis of the 5W + 1H questions. They answered all questions by introducing their self-adaptation taxonomy, although the "why" question was not answered because of the nature of SAS. Elhabbash et al. [9] presented another survey on self-awareness in software engineering. Although their paper focuses on the field of software engineering, it covers many important questions regarding the definition of self-properties and the engineering approach to developing self-awareness software. Most surveys, including those mentioned earlier, focus on SAS. To the best of our knowledge, we are unaware of any survey paper focusing on self-adaptive IoFT. However, some literature partially addresses self- properties in IoFT. For instance, Abdulrahman et al. [10] discuss self-optimization in resource management within a federated ecosystem. This study demonstrates the reasons and goals behind using this approach across various domains.

1.2 Contributions

Although there are systematic reviews of both SAS and FL, most studies treat the two topics separately. Furthermore, a thorough study of self-adaptive IoFT is yet to be conducted. Therefore, our contributions can be summarized as follows.

- We have thoroughly evaluated the literature related to the implementation of self-adaptive IoFT systems.
- We briefly define SAS, FL, and IoT and their unique characteristics and relationships.
- We introduce our taxonomy, highlighting the main properties of self-adaptive IoFT systems.
- We cover aspects of self-adaptive IoFT systems, such as context, motivations, implementation, applications, challenges, and future direction.
- We introduce a conceptual architecture with feedback loop for self-adaptive IoFT, which is demonstrated through anomaly detection (AD) in smart home environments.

Table 1: List of Abbreviations

Phrase	Acronym	Phrase	Acronym
Federated Learning for IoT Application	IoFT	Research Question	RQ
Federated Learning	FL	Inclusion Criteria	IC
Internet of Things	IoT	Exclusion Criteria	EC
Industrial Internet of Things	IIoT	Transmission Control Protocol	TCP
Self-adaptive System	SAS	User Datagram Protocol	UDP
Machine Learning	ML	Internet Protocol	IP
Reinforcement Learning	RL	Remote Procedure Call	RPC
Neural Network	NN	Network Functions Virtualization	NFV
Deep Learning	DL	Software-Defined Networking	SDN
Anomaly Detection	AD	Virtual Machine	VM
Artificial Intelligence	AI	Systematic Literature Review	SLR
Independent and Identically Distributed	IID	Stochastic Gradient Descent	SGD

Table 2: Previous Survey Comparison

|Paper	|Year	|Method	|Domain	|Focus	|Studies		|Self-adaptation system					
					considered	analysed	definitions	motivations	methods	evaluation	applications	IoFT
[6]	2008	Adh	AC	C|H|O|P	N/A	N/A	Ex	✓	✓	x	✓	x
[7]	2009	Adh	SE	C|H|O|P|A|D|CA	N/A	N/A	Ex	✓	✓	✓	✓	x
[8]	2015	Adh	AC	C|H|O|P|A|D|CA	N/A	N/A	Ex	✓	✓	✓	✓	x
[9]	2019	SLR	SE	H|P|A|D|CA	865	74	Ex	✓	✓	✓	✓	x
[11]	2022	SLR	CPS	H|P|A|D|CA	266	30	Ex	✓	✓	✓	✓	x
[10]	2021	Adh	FL	O	N/A	N/A	x	P/C	P/C	P/C	✓	✓
This Paper	2024	SLR	FL	C|H|O|P|A|D|CA	290	20	Ex/Im	✓	✓	✓	✓	✓

C: self-configuring **H**: self-healing **O**: self-optimization **P**: self-protecting **A**: self-awareness, **D**: self-adaption **M**: self-managing **L**: self- learning, **CA**: Context-awareness **SLR**: systematic literature review **Adh**: ad hoc **Ex**: Explicit **Im**: Implicit **IoFT**: Internet of Federated Things, **FL**: Federated learning **SE**: Software engineering **AC**: Autonomic computing **CPS**: Cyber–physical system **P/C**: partial coverage, **N/A**: not applicable

1.3 Paper Structure

The remaining sections of this paper are structured as follows: Section 2 introduces the background knowledge for the rest of this paper by describing fundamental topics and terminologies. Section 3 highlights the research methodology used in this study. Section 4 presents the definition, motivation, implementation, evaluation, and application of self-adaptive IoFT. In this section, we also introduce the feedback control loop for self-adaptive IoFT and demonstrate how it works for self-adaptive IoFT for AD in smart homes. At the end of this section, we discuss the challenges and future research directions for self-adaptive IoFT. Finally, we conclude this work in Section 5.

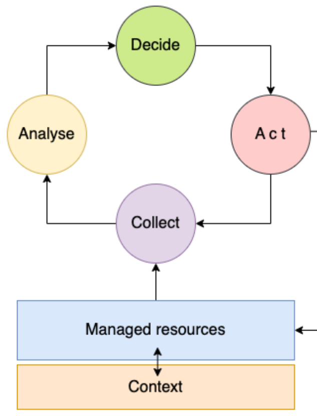
2 Background

2.1 Self-adaptive Systems

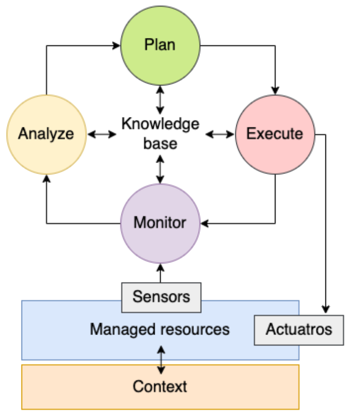
SAS is designed to deal with the increasing complexity of systems that face uncertain aspects at runtime. In the early 2000s, the research community introduced the concept of SAS [7] [12] [13], which describes a system that can configure itself using various mechanisms to preserve system quality within an unstable environment. Uncertainties can take different forms, such as introducing a device to the current network, adapting a new user's behavior to smart places, and exceeding the threshold due to an unknown event in cloud resources. Therefore, a system that can adapt to the dynamic environment is crucial. The core design philosophy of an SAS is to distinguish between the adaptation logic that maintains or enhances certain system qualities and the managed resources that execute the domain-specific functions of the application. Therefore, the research community has provided a myriad of proposals and system architecture in the last two decades to design an SAS. We briefly discuss some popular conceptual designs for developing and using SAS in the literature.

2.1.1 Autonomic Feedback Loop

Driven by autonomic communication, which focuses on communication, networking, and distributed computing paradigms, Dobson et al. [14] proposed an autonomic control loop, as shown in Figure 1a to enable self-adaptation to autonomic communication. They stated that the system begins gathering information from various sources, such as



(a) Autonomic feedback loop



(b) MAPE-K model

Figure 1: Conceptual designs of autonomic feedback loop and original MAPE-K model

network traffic, sensor readings, user context, and application requirements. The data will be analyzed using different techniques, such as decision theory and risk assessment. This analysis will help create a model representing the system's current state, which will be used later to make decisions. Once the decision-making stage is complete, the results are moved to the action stage, which provides instructions to the system administrators or records strategies for the next control cycle.

2.1.2 Traditional and MAPE-K Feedback Loops

The MAPE feedback control loop is a well-recognized engineering approach to enable self-adaptation, which also follows the four computation components monitor–analyze–plan–execute [12]. The monitoring stage initiates the control loop. This stage gathers relevant data from the environment representing the current state of the system. After that, in the analysis stage, the system should analyze the output of the previous monitoring stage using various methods to organize and reason the information; we will explore a few methods later in Section 4.3. In the planning stage, a set of actions to adapt the managed resources is defined, which allows a reactive response to any event that may evolve over time. These actions and responses will be executed in the last stage to enable adaptation within the system. Finally, a MAPE-K feedback loop improves MAPE by adding a knowledge base to share the data across all computation components in Figure 1b.

2.1.3 Architecture-based Self-adaptation

Oreizy et al. [15] introduce one of the early works on architecture-based self-adaptation. Unlike the other self-adaptation approach, Oreizy's method includes a two-phase model for an SAS: evolution and adaptation management, which include key processes that help attain the goals and objectives of SAS. First, the evolution management phase has several components to make changes and constantly collect observations. This phase minimizes changes and accidental errors during model execution. Evolution management mainly deals with applying change over time while minimizing the risks associated with the process. Second, the adaptation management phase examines system behaviors to react accordingly and identify appropriate adaptations. The essential practices of this phase include evaluating, monitoring, and planning changes. The model mainly addresses the issue of inconsistencies and errors that occur unplanned.

2.1.4 Rainbow Framework

Garlan et al. [13] introduce the rainbow framework, which integrates the architectural model of a system in its runtime system in contrast to conventional applications of software architecture as a purely design–time artifact. Developers of self-adaptation capabilities specifically use the software architectural model of a system to track and analyze the system. Therefore, the rainbow framework design consists of two layers: system and architecture. Separating the system design reduces the development cost and increases the usability of the framework because it can be used with any system, such as a legacy system. Rainbow realizes self-adaptation using the model manager to support the constraint evaluator, which provides reasoning features to the current system and identifies its behavior. The reasoning features support planning for the following action through the adaptation engine, which is later translated into action via the adaptation executor.

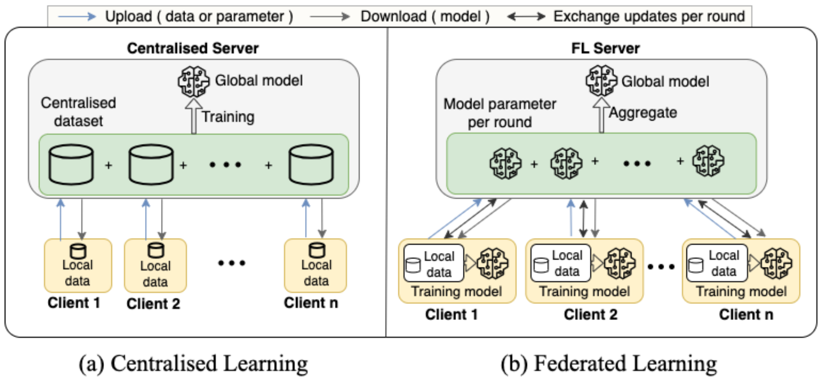


Figure 2: Architectural design for (a) centralised learning vs. (b) federated learning

2.1.5 Common Features of SAS

We have listed some approaches in the literature to enable self-adaptation systems. These self-adaptation techniques clearly share some common features, which can be summarized into three main observations. First, a crucial feature is the separation of the control logic and managed resources. This separation enhances the usability and portability of these approaches because it allows easy adaptation to new or existing systems. Moreover, it reduces the complexity of the system by splitting it into small subsystems. From a system design perspective, the separation allows engineers to focus more on system operation and quality maintenance rather than on the system interface with the adaptation logic component. Second, reducing the number of adaptation logic and managed resource interactions is one of the main objectives. Consequently, the system permits costly operations on the adaptation logic component and returns feedback as a single or set of actions to be applied to managed resources or notified to the system administrator. Third, the main structure of the adaptation cycle in these approaches contains several steps. Each is responsible for a specific task using data from the previous step. However, all these steps use some form of state management, sharing common variables and system configurations to synchronize the data across all components.

2.2 Federated Learning

2.2.1 Definition

Several models are used for ML in the IoT, including centralized and decentralized approaches, each with its advantages and disadvantages [16]. The FL paradigm for IoT is a new addition to these methods. FL applications can be seen in various fields, including smart city [17], health care [18], recommendation systems [19], edge network [20], electric grid [21], vehicular ad hoc network [22], and blockchain [23]. In contrast to centralized ML approaches, FL inherently enhances security and privacy by keeping data localized at the edge. In this framework, data generated on edge devices are used for local training of ML models rather than being transmitted to a central server. Consequently, only model parameters are exchanged between the edge devices and the cloud server, as shown in Figure 3. Therefore, the FL process generally includes three main steps.

- Step 1: A federated server initiates the training task and creates an initial global ML model. The federated server also determines the list of contributed clients, the number of rounds, and the aggregation process for all incoming parameters.
- Step 2: After receiving the model from the federated server, the federated clients train the model using their local data. After completing this training phase, they transmit the newly refined model, which is returned to the federated server.
- Step 3: After receiving refined models from the federated clients, the federated server aggregates the received model parameters to create an updated global model. Subsequently, the global model was redistributed to all participants, initiating another training round.

These processes (i.e., those mentioned in steps 2 and 3) will continue until the federated server reaches the maximum number of rounds or convergence. In the literature, the terms specialization and generalization are used as alternative terminology for communication. After convergence, additional rounds may be required if a new federated client joins or new data are collected. Determining how federated servers aggregate the received model into a global model is critical. Numerous FL aggregation approaches have been reported in the literature. The most common method is

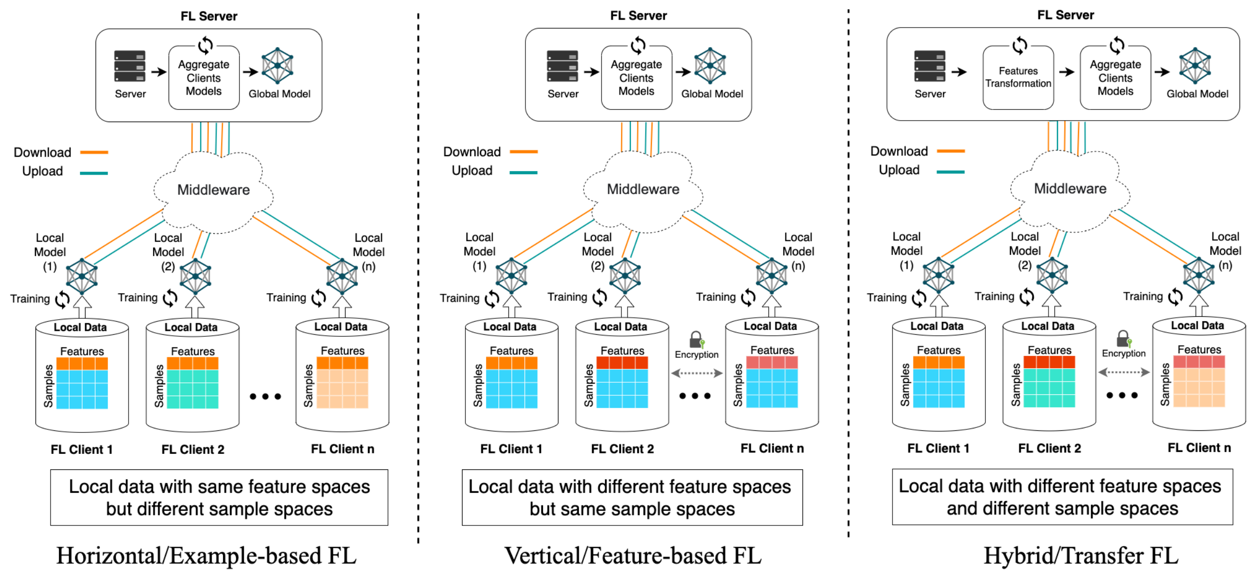

FL Server
Server
Aggregate Clients Models
Features Transformation
Global Model
Download
Upload
Middleware
Local Model (1)
Local Model (2)
Local Model (n)
Training
Local Data
Features
Samples
Encryption
FL Client 1
FL Client 2
FL Client n
Local data with same feature spaces but different sample spaces
Local data with different feature spaces but same sample spaces
Local data with different feature spaces and different sample spaces
Horizontal/Example-based FL
Vertical/Feature-based FL
Hybrid/Transfer FL

Figure 3: Data partitioning types in FL

federated averaging denoted by FedAvg [24]. Thus, the proposed algorithm attempts to reduce the loss function and achieve convergence by averaging the model weights received from the client, thereby making it the baseline of FL. The FedSGD algorithm is commonly used in deep learning because it aggregates gradients and performs a step of gradient descent [24]. However, FedAvg distributes updated weights rather than gradients, which allows local clients to perform multiple batch updates on local data. If all local clients have the same start model parameters, averaging the gradients in FedSGD is equivalent to averaging the weights in FedAvg .

2.2.2 Data Partitioning

An essential stage in designing an FL system is data distribution analysis over samples and feature spaces. Therefore, the FL system can be typically categorized as horizontal, vertical, or hybrid FL.

Horizontal/Example-based FL The datasets of different participants are different or have a small intersection in the sample space; however, the feature space is identical. Horizontal FL is a helpful technique for partitioning data, particularly when multiple participants attempt to enhance ML model performance on similar tasks using FL. This type of horizontal data partitioning is commonly used in FL research. Because local data are in the same feature space, participants can train their local models using the same ML model architecture and local data. FedAvg was therefore used in the horizontal FL to average all local models. A typical application of wake word detector [25] and a word prediction [26].

Vertical/Feature-based FL The data space between participants is the same but differs in terms of the feature space. For example, if we want to create a generalized model that works across fields, we can combine different features of different sectors to train a single global model. Therefore, this type of data partitioning requires a different approach to achieve the FL task. For example, entity alignment techniques can acquire participant knowledge and identify overlap samples between participants [27]. Cheng et al. [28] proposed a vertical FL method that allows participants to train gradient-boosting decision trees collaboratively without loss of information. Their approach finds commonalities between different FL client data that join the decision tree process while preserving the privacy of the FL client.

Hybrid/Transfer FL This is a combination of both horizontal and vertical FL data partitions. Another name used in the literature is federated transfer learning. The main reason for introducing this hybrid data partitioning is that if we have two or more FL clients with overlapping sample data and features, the FL will perform poorly because of the heterogeneity of the data and feature spaces. An example of an application that could benefit from hybrid FL is a marketing company that collects customer data to launch marketing campaigns in different countries. Because of cultural variations, thoughts, and beliefs, some countries may have more features that work well to predict a marketing campaign's success than others. However, there is a small intersection in the feature space, such as age interest, which can be generalized across countries. Furthermore, because of the different geographic locations of the two marketing campaigns, the overlap of the data sample was negligible. Thus, hybrid FL attempts to benefit from both data partitioning techniques by transferring learning [29] to another FL client. Liu et al. [30] provide a secure transfer FL system that uses shared features and samples to learn a representation of an FL client.

2.2.3 Federation Scale

FL scaling classification encompasses cross-device [31] and cross-silo [32] applications based on the number of participants and the volume of data distributed within the federation.

Cross-silo Cross-silo FL is typically used in cross-domain FL such as banking or medical or for different geographically distributed data centers. Cross-silo FL leverages extensive datasets from some FL clients to enhance the training of the global model. Therefore, the FL client count is small, typically encompassing companies and organizations. An example of a cross-silo is YouTube, which shows targeted advertisements by training models using data collected from different geographical locations and storing them in the nearest Google data center. The data were configured, and the model was trained using sufficient computational resources. Because of the flexibility of cross-silo FL, data partitioning can be either example (horizontal) or feature (vertical) based. Moreover, many FL clients are incentivized to train a model using all their data in the cross-silo context. However, they cannot exchange their data directly because of secrecy, regulatory limitations, or even when they cannot organize their data to meet the initial data requirements of FL.

Cross-device There are potentially vast numbers of FL clients; only a tiny fraction are available at any given time. The type of FL client can vary because of the heterogeneity of devices such as smartphones and IoT devices. Most of these devices have limited computational resources, making them ineffective for training. In this case, the federated server must be able to manage all revised local models to aggregate a global training model. For example, Google has proposed an FL-based keyboard suggestion using end-user devices to locally train a keyboard suggestion model [33]. Similarly, Apple uses cross-device settings to train Siri to recognize various voices [34]. Because of the nature of the cross-device FL setting, it is impossible to directly address and index participating clients. This limitation reduces reliability, unlike cross-silo, where each client can be identified easily to make it accessible [32].

2.2.4 ML Techniques

FL in IoT uses several ML techniques. Technique selection is primarily based on the goal to be achieved and the type of available data. Many domains use different ML techniques in addition to SAS. Supervised, unsupervised, and reinforcement ML algorithms are common approaches to building SASs. Some examples of ML techniques are based on Bayesian theory [35][36][35], clustering [37], fuzzy learning [38], genetic algorithms [39] [40], neural networks [41][42], and decision trees [43][44][45]. Saqutri and Lee [46] provide a comprehensive review of ML for SAS. Detailed qualitative and quantitative syntheses of 231 studies that reflected state-of-the-art federated ML were analyzed recently [47].

2.2.5 FL Frameworks

There are many frameworks used in the FL research community. These frameworks were primarily created to be implemented on real-world systems, which decreases the entrance barrier since it does not require substantial knowledge of different computing disciplines. Some well-known frameworks are PySyft [48], FedML [49], LEAF [50], Flower [51], Clara [52], PaddleFL [53], Open FL [54], TensorFlow-Federated [55], FATE [56]. Burlachenko et al. [57] present FL_PyTorch, an FL simulator, to deduce the preliminary required to implement FL without expert knowledge.

2.2.6 Relationship between FL and IoT

Several limitations regarding the current IoT ecosystem paradigm for implementing ML tasks should be noted. These limitations include lack of data availability; violation of end-user privacy; high communication costs; heterogeneity of IoT devices; and challenges in the scalability, availability, and reliability of ML functionality [58]. FL plays an essential role in addressing the limitations of ML in the IoT domain. FL tackles the lack of data availability by allowing several parties to join the collaboration and share their model parameters trained within other parties' local data. The conventional approach for training ML in the IoT domain requires all benefited parties to share their data with a central server to perform model training. However, many organizations, such as health care, refuse to accept this idea of collaboration because of the high privacy violation of sharing end-user private data. To preserve privacy, FL discourages all joined parties from sharing their local data; instead, they may share the model's parameters such as gradient and weight. Moreover, the communication costs associated with offloading local data to the cloud are critical for traditional ML in IoT systems. Additionally, the heterogeneity of the IoT introduces new challenges because of the capabilities of

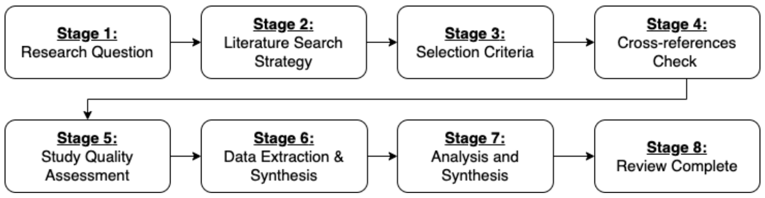


Figure 4: Research methodology stages [60].

IoT devices, which can affect the overall training process and the performance of the global model. The architecture of FL provides a novel solution for both communication costs and the diversity of IoT devices by taking advantage of the distributed nature of FL and the type of data required from joined parties. Each FL client is only required to share a small amount of data, process less payload, and improve network bandwidth. Additionally, the aggregation process of the FL server can be efficiently used to manage the diversity of IoT devices by leveraging different incentive mechanisms to reward FL clients based on their contributions, such as data quality and quantity. One of the biggest challenges regarding the scalability, availability, and reliability of conventional ML systems in the IoT domain is that all training is conducted on a single centralized server, making it more vulnerable to threats. Additionally, as IoT devices grow, handling scalability becomes overwhelmingly complex. In the FL architecture, the FL client is responsible for training the global model with its local data, and the FL server is responsible for global model initiation and aggregation of all FL client contributions. FL clients need not join the training round, which can be disconnected anytime. However, they will lose the benefit of FL by obtaining the most updated model. Some FL architectures, such as FL with the incentive aggregation technique, are designed to manage many FL clients and measure their contributions. Finally, the primary objective of adapting IoFT is to overcome the limitations of conventional ML systems when applied to continuously changing environments.

3 Review Methodology

The advantage of a systematic literature review (SLR) is that it is well- known for evaluating papers that fit the prespecified eligibility criteria [59]. Moreover, the SLR is concerned with identifying, analyzing, and assessing research findings relevant to specified research questions. To conduct this study, we performed both manual and automatic searches. We investigated the implementation of self-adaptive IoFT in edge, fog, and cloud computing scenarios. We used backward and forward reference search methods to identify the most relevant results. We followed the research methodology outlined in Figure 4.

- *Stage 1 : research questions*: We intend to present a comprehensive and structured overview of all significant self-adaptive IoFT articles related to the following research questions.
  - RQ1: What is the definition of self-adaptive IoFT systems?
  - RQ2: What are the characteristics of self-adaptive IoFT systems?
  - RQ3: What is the feedback loop architecture for self-adaptive IoFT systems?
  - RQ4: What are the primary motivations for using self-adaptive IoFT systems?
  - RQ5: What are the technical considerations for implementing self-adaptive IoFT systems?
  - RQ6: How are self-adaptive IoFT systems evaluated?
  - RQ7: What is the reality of self-adaptive IoFT systems?

  RQ1 addresses the definition of self-adaptive IoFT in the literature and its different interpretations in Section 4.1. RQ2 is motivated by the need to define and characterize self-adaptive IoFT capabilities. Additionally, we summarize several existing taxonomies used in the literature to identify the primary characteristics of self-adaptive IoFT in Section 4.2. RQ3 addresses the adaptation logic and conceptual architecture of feedback loop control for self-adaptive IoFT in Section 4.3. RQ4 explains the motivation for using self-adaptive IoFT in real-world applications in Section 4.5. To conduct a thorough analysis, RQ5 examines various engineering approaches for developing a self-adaptive IoFT system in Section 4.6. The purpose of RQ6 is to identify various evaluation techniques, criteria, and metrics proposed in the literature to assess the performance and reliability of the IoFT in Section 4.7. Finally, RQ7 demonstrates the reality of self-adaptive IoFT by examining real-world applications and domains that use this paradigm in Section 4.8.
- *Stage 2 : literature search strategy*: The literature search strategy begins by identifying data sources and a search query. The study's search strategy is based on an automated search of multiple globally known databases and indexing systems such as Google Scholar, Scopus, ScienceDirect, AMC Digital Library, IEEE

Table 3: Search Queries

Database or Indexing Services	Search Queries
Google Scholar	intitle:"federated learning" OR intitle:FL AND IoT OR "Internet of Things" AND intitle:self-* AND -intitle:review AND -intitle:survey AND -intitle:systematic
Scopus	TITLE-ABS-KEY(("federated")AND(self*) AND ("IoT" OR "Internet of Things")) AND PUBYEAR > 2015
ScienceDirect	("federated") AND ("IoT" OR "Internet of Things")
ACM	[Abstract: "federated"] AND [Abstract: self*] AND [Publication Date: (01/01/2016 TO 31/12/2023)]
IEEE	("Abstract":"federated") AND ("Abstract":"self*") AND ("Abstract":IoT OR "Abstract":"Internet of Things")
Springer Link	"federated learning" OR "Internet of Federated Things" AND self-* AND (review OR survey OR systematic)

Xplore, and SpringerLink to collect relevant information from published sources. We searched for keywords in the lists of databases and scientific citation indexing services. Our automatic search technique focused on the keywords “Federated learning,” “Internet of Federated Things,” “Autonomic computing,” and “Context-aware.” Because of the ambiguity surrounding the definition of self-adaptation, we used a wildcard search to obtain all relevant results containing terms such as “Self-configuration,” “Self-learning,” and “Self-optimization.” Table 4 shows the automatic search results before snowballing.

- *Stage 3 : selection criteria*: We conducted two rounds of study selection against the findings of automatic searching to determine the primary study. We carefully filtered the papers in the first round according to their titles, abstracts, and keywords. Additionally, duplicate data were eliminated from various sources. In the second round, we filtered the primary studies using well-defined inclusion (IC) and exclusion (EC) criteria.
  - IC1: Google first proposed FL in 2016 as an alternative setting for centralized ML approaches [24]. Therefore, we limited our search to papers published after January 1, 2016, demonstrating self-adaptive IoFT.
  - IC2: This study demonstrates an ML-based approach for IoFT and its properties. However, any study that incorporates a self-adaptive technique that explicitly mentions the IoFT, such as federation in cloud computing, software, or blockchain, will be included.
  - IC3: The study discusses self-logic in general. Because this study focuses on IoFT, we include only research that applies the self-adaptive IoFT environment for self-adaptation.
  - EC1: The study should not be an abstract or limited to one or two pages. These studies are excluded because they often need more information.
  - EC2: The study should not use self-adaptation in contexts other than FL and IoT. This study does not contribute to answering the main research question because we focus on the IoFT.
  - EC3: A study that focuses on theory without proof of concept will be excluded because this study must meet our quality assessment criteria.
- *Stage 4 : cross-references check*: To ensure that no relevant research is overlooked, we use the cross-referencing methodology and identify potentially relevant papers using the “snowballing” search strategy. We accomplished this by recording the references listed in the “References” section of each primary study [61] [59].
- *Stage 5 : quality assessment criteria*: Kitchenham et al. [61] [62] stated that the quality of research is related to its ability to minimize bias and maximize internal and external validity. The primary studies were evaluated according to predefined quality assessment criteria. Moreover, we used checklists provided by reference [62]. We include each paper that has defined the problem statement and contribution, presented background and context, provided a clear description of the research method and evaluation, and reported the findings.
- *Stage 6 : data extraction item*: We review all selected primary studies to gather data to help answer the research questions. Table 5 describes the data items to be retrieved and their corresponding research questions.
- *Stage 7 : analysis and synthesis*: Stages 3, 4, and 6 of the research method used in this study examined the analysis and synthesis of research publications. Data items defined earlier in stage 6 were extracted and recorded in a spreadsheet for each study. Additionally, we used various software tools, such as NVivo and Excel, to analyze and visualize the findings of the selected primary studies.
- *Stage 8 : reporting the review*: At this stage, we present the results of our analysis of the 20 selected primary study data after the filtering process conducted in all previous stages and the answers to our research questions.

Table 4: Search Results Before Snow-balling

Google Scholar	IEEE	ACM	ScienceDirect	Scopus	Springer Link
64	21	50	23	104	28

Table 5: Data Item Collection Form

Data item	Description	Relevant QR
Title	for paper's reference	Documentation
Year	The publication year of the primary study	Documentation
Publication source	The Publication metadata and type of primary study	Documentation
Definition	The definition of self-adaptive IoFT	RQ1
self-* property	The properties of self-adaptive IoFT	RQ2, RQ3
self-* taxonomy	The self-* taxonomy on different in the field of a self-adaptive system	RQ2, RQ3
Feedback loop control	The feedback loop architecture concept	RQ3
Motivation	The primary motivations for using self-adaptive IoFT	RQ4
Technical aspect	The technical considerations for implementing self-adaptive IoFT s	RQ5
Assessment method	The various evaluation techniques, criteria, and metrics for assessing self-adaptive IoFT	RQ6, RQ7
Applications	The real-world applications and domains that employ the self-adaptive IoFT	RQ7

4 Self-adaptive IoFT

4.1 The Definition of Self-adaptive IoFT

To define a self-adaptive IoFT, we first need to understand two terms: context and context-aware systems. These terms form the foundation of every SAS.

4.1.1 Context and Context-aware Computing

The primary source of data generation is the context. However, in the literature, many researchers have defined the term "Context" differently based on their domain expertise. For example, Brown [63] defines context as the components of the user's surroundings that are known to the computer. Salehie et al. [7] extend Brown's definition; thus, the context is everything in an operating environment that influences the system's attributes and behaviors. However, the most accurate definition was proposed by Abowd et al. [64]. These authors refer to context as:

> ”*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*”

Abowd et al. [64] state that the context comprises three entities: people (individuals and groups), places (buildings and offices), and things (sensors and actuators). These entities are characterized by several attributes: identity, location, status, and time. Abowd et al. [64] define these categories as the primary context. In contrast, the secondary context can only be obtained from the primary context with additional information, such as weather based on the user's location. However, in IoT applications, this definition is only partially applicable. Perera et al. [65] argue that the categorization scheme proposed by Abowd et al. [64] did not consider ordinary IoT contexts. For example, calculating the distance between two GPS sensors involves processing the locations obtained using both sensors. This type of data processing commonly occurs in IoT applications because most data interpretations are derived from a group of sensor readings. The definition proposed by Abowd et al. [64] does not address this common scenario in IoT applications. Therefore, Perera et al. [65] extend the original definition to include an operational perspective. Their definition highlights the challenges in data acquisition in IoT applications and provides a conceptual perspective to understand the relationships between different contexts. Our study adopts the categorization scheme proposed by Perera et al. [65] because it is better suited to the IoT paradigm and captures both the operational and conceptual attributes of context-aware IoT applications. Context-aware computing is derived from the desire to use contextual information. Numerous viewpoints on how systems should consider context have been provided [65] [64] [63] [7]. However, the primary purpose of context-aware computing is to evaluate the context and respond to dynamic environment changes to meet specific goals based on relevant information. In the IoT setting, context-aware systems connect context information to sensor data to provide insight for interpretation.

Table 6: Definitions of Self-* Property in IoFT System

Study	Express	Property	Definition
[69]	Implicit	self-organization	*The ability of federated learning clients to autonomously enhance the uploaded weight of parameters to the federated learning server.*
[70]	Implicit	self-organization	*The ability of federated sensor network to autonomously detect resources, optimise the network routing protocol and algorithm*
[71]	Implicit	self-learning	*The ability of federated learning to learn the device's type and anomalies autonomously.*
[72]	Explicit	self-organization	*The ability of federated learning to autonomously create different collaboration schemes to identify the heterogeneity hidden in the federation.*
[73]	Implicit	self-organization	*The ability of federated learning to autonomously formalities device clusters, join devices, and allocate resources.*
[74]	Explicit	self-adaption	*"In the context of self-adaptation, the roles are monitor, analyze, plan and execute over a shared knowledge "*
[75]	Explicit	self-adaption	*"FedML solutions configure the system, that is, set its parameters, and allocate resource dynamically"*
[62]	Explicit	self-awareness	*"we refer to a SASO system S as a collection A of autonomous subsystems ai that are able to adapt their behaviour based on self-awareness of the internal and external conditions"*
[76]	Explicit	self-attention	*"By using the self-attention mechanism, we can optimize both the server-to-client and the client-to-client parameter divergence and increase the model's performance to Non-IID data. "*
[77]	Explicit	self-revealing	*"The self-revealing mechanism of the contract theory approach enables workers to be rewarded based on their specific types even in the presence of information asymmetry, i.e., when the worker types are not known by a model owner."*

4.1.2 Self- IoFT Systems

An early definition of the term "self" emerged in the late 1890s within psychology, where Baker [66] described "self" as a process of identification that marked the beginning of scholarly exploration. Goffman [67] further extended this conceptualization of "self" in sociology, illustrating "self" as a dynamic entity influenced by varying circumstances and contexts. The transition from these foundational ideas to the technological domain was marked by IBM's introduction of "autonomic computing," aimed at developing systems capable of self-management [12]. This manifesto led to the emergence of "self-" systems, which encompass behaviors such as self-configuration, self-optimization, self-healing, and self-protection. The concept of "self-" systems has evolved rapidly, prompting efforts to define it broadly, despite the lack of a universally accepted definition, as highlighted in [9][68]. The literature presents two opinions on the definition of "self-." Initially, the definition is influenced by the author's perspective and the context in which it is applied. Alternatively, it is shaped by the domain that adopts the "self-" method. This variation underscores the adaptability of "self-" concepts across different scientific fields, including the IoFT, where such systems play crucial roles. Table 6 shows explicit and implicit definitions of "self-" in IoFT systems, as found in primary studies.

4.2 Characteristics of Self-adaptive IoFT

There has been a significant effort in the literature to develop various taxonomies to identify key characteristics of SAS over the years. A notable contribution to this field is the comprehensive empirical analysis conducted by Krupitzer et al. [8], which spans early studies from the 2000s. This study aimed to establish a uniform taxonomy for self-adaptation by addressing the 5W + 1H questions, a concept first introduced by Salehie et al. [7]. The taxonomy presented by Krupitzer et al. is summarized into five dimensions: time, level, technique, reason, and adaptation control, providing a general overview of the SAS landscape. Other researchers, such as Andersson et al. [78], have proposed alternative dimensions that view self-adaptation through system goals, triggers, mechanisms, and adaptation outcomes. Brun et al. [79], who proposed five dimensions specifically for software engineering further refined this perspective: adaptation targets, effects, actions, states, and the environment. Although these contributions are pivotal, they predominantly reflect the viewpoints and requirements of the software engineering community without directly addressing the unique aspects of the IoFT. Therefore, we formed the IoFT taxonomy in Figure 5, drawing inspiration from the aforementioned studies. The multidimensional taxonomy is organized to encapsulate the distinctive features of IoFT, providing a novel framework that complements existing research while addressing the specificity of IoFT applications.

4.2.1 Time

In an ideal scenario, proactive rather than reactive adaptation is preferred to ensure performance continuity. Proactive adaptation requires accurate prediction capabilities that requires ongoing monitoring and sophisticated learning techniques [8]. In contrast, reactive adaptation begins after the recognition of a change requirement and responds to irregular environmental patterns. Therefore, the adaptation process in a reactive model is initiated by detecting these

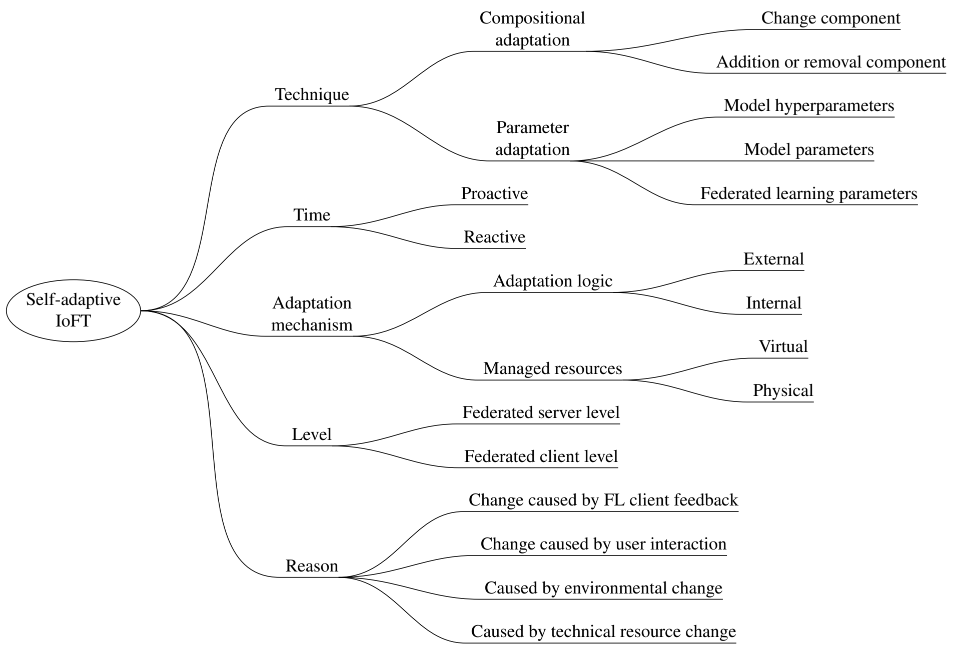


Figure 5: A taxonomy of characteristics for self-adaptive IoFT

irregularities. Conversely, proactive adaptation seeks to anticipate potential environmental changes before performance degradation occurs, focusing on predicting and preparing for future environmental shifts [7] [8].

4.2.2 Reason

Adaptation is often a response to change. Consequently, the type and effect of the change must be defined to determine whether a response is required. Additionally, it is essential to identify the cause of the change and define criteria for an appropriate method to respond to that change. The main reasons for IoFT adaptation are (1) in technical resource changes, (2) environmental changes, (3) changes due to user interactions, and (4) downgraded feedback from FL clients. First, a change in technical resources includes all tangible (IoT device and hardware components failures) and intangible (software failure, FL server model divergence, FL client aggregation errors, and unstable network connection) assets. Second, the environment changes when the context of the federated clients changes from one context to another (smart home, building, and manufacturing). Third, a change due to user interactions can occur by either altering user behavior using IoT devices (fire alarm leading to evacuation) or changing user preferences to interact with IoT devices and sensors. Finally, the FL server may trigger the adaptation strategy if feedback from the FL client does not meet the current service requirement (lack of availability to join the training, poor model training due to data availability, or model poisoning attack).

4.2.3 Level

The implementation of adaptation can exist at different levels. Various existing taxonomies attempt to answer the following questions: “which layer of the system can be changed” [7] and “where do we have to implement changes” [8]. These questions have been addressed in the literature. However, IoFT does not explicitly declare the adaptation levels. There are two levels in IoFT: the FL server and client. They provide a high-level abstraction of where managed resources and adaptation logic exist . An FL server has many adaptation mechanisms, such as client selection, global model selection, and model aggregation optimization. Adaptation also occurred in the FL client, where different actions, such as local model optimization, automatic sensor configuration, and automatic delivery of actuation orders, can be presented.

4.2.4 Technique

The literature uses several adaptation techniques from different fields. McKinley et al. [84] mentioned two approaches for adaptation in software engineering: (1) parameter and (2) compositional adaptation. Parameter adaptation is the most straightforward technical approach because it can be achieved by simply identifying the system’s parameters

Table 7: FL Optimisable Parameters in FL

Category	Parameter/Algorithm	Definition/Parameter Details
Global FL Parameters	fraction_fit	The proportion of randomly selected clients participating in each training round reflects the system's strategy.
	fraction_evaluate	The proportion of clients selected for model evaluation after training.
	min_fit_clients	The minimum number of clients required to proceed with a training round.
	min_evaluate_clients	The minimum number of clients needed to validate the model's performance.
	min_available_clients	The minimum number of clients that must be online and available for the FL system to function.
	accept_failures	A policy determining whether training rounds that experience client failures are accepted or discarded.
	initial_parameters	The initial global model parameters serve as a baseline for subsequent iterations.
Aggregation Parameters	FedAvg [24]	(c) Number of clients participating in each training round, (e) Number of training epochs each client makes over its local dataset on each round, (b) The local mini-batch size used for training in client side
	FedSGD [24]	(w) Model weights on communication selected round , (c) Number of clients participating in each training round, (e) Number of training epochs each client makes over its local dataset on each round, (b) The local mini-batch size used for training in client side, (eta) The learning rate
	FedAdam, FedYogi [80]	(eta) Server-side learning rate, (eta_l) Client-side learning rate, (beta_1) Momentum parameter, (beta_2) Second moment parameter, (tau) Controls the degree of adaptability
	FedAdagrad [80]	(eta) Server-side learning rate, (eta_l) Client-side learning rate, (tau) Controls the degree of adaptability
	FedProx [81]	(proximal_mu) The weight of the proximal term used in optimization
	FedTrimmedAvg [82]	(beta) Fraction to cut off of both tails of the distribution
	q-FedAvg [83]	(eta) learning rate, (q) Tune the amount of fairness

and changing them according to the adaptation policies. Compositional adaptation is a dynamic approach to changing algorithms or system components to reduce performance degradation. Therefore, we leverage these two adaptation techniques for self-adaptive IoFT. In self-adaptive IoFT, parameter adaptation must consider three general FL system design tiers: (1) FL server parameters and ML model (2) parameters and (3) hyperparameters. The FL server parameter adaptation achieves adaptation by adjusting the global configuration settings of the FL server and aggregator parameters, which vary depending on the aggregation algorithm. Table 7 shows the literature's most used aggregation algorithms and their configurable parameters. The ML model parameters allow the changes to the ML algorithm parameters to obtain a better starting model. The adaptation can occur on the FL server or client side depending on the predefined adaptation strategy. Changing model parameters may require reinitializing the FL system to implement the changes. Unlike ML model parameters, ML model hyperparameters enable changes at runtime in any part of the FL system, including the FL server and client. Two methods can archive compositional adaptation: (1) change components by replacing the ML algorithms or adding more models in the case of ensemble ML and 2) add or remove components, such as including or excluding FL clients, based on their contributions and effectiveness to global model convergence. Finally, the adaptation triggers are based on specific predefined policies or criteria where adaptation is required to prevent performance loss or to obtain better overall performance.

4.2.5 Adaptation Mechanism

The adaptation mechanism is the core element of any SAS [8], [7]. [8] highlights that adaptation control can be compressed into two main components: managed resources and adaptation logic. The same components are also applied to the self-adaptive IoFT domain, but with different interpretations of managed resources and adaptation logic.

Managed resources Managed resources in the IoFT ecosystem encompass physical and virtual resources. The physical resources of FL clients vary according to their specific domain, such as motion sensors in smart homes or heart rate monitors in health care. Despite their diversity across domains, these resources function universally as sensors, actuators, small devices, and gateways. For example, a sensor translates physical events into electrical signals, whereas an actuator executes physical actions based on electrical inputs. Other examples of physical resources are small devices and IoT gateways. Small devices, such as mobile phones or Raspberry Pi and any control system device with limited computational power . A gateway connects the IoT devices to the internet, facilitating data transmission between the FL client devices and the FL server. In contrast, virtual resources represent the capabilities of these physical resources, such as the accuracy of sensor data or the storage capacity of devices. These intangible factors are crucial to the efficiency of IoFT ecosystems. In the FL server, resources are represented differently, which aligns better with cloud and fog computing paradigms. Physical resources refer to devices hosting FL orchestration mechanisms, ranging from PCs to virtual machines in the cloud. Virtual resources in the FL server contain performance metrics such as computing power and autoscaling capabilities.

Table 8: The Evaluation of Self-adaptive IoFT Characteristics Taxonomy Against The Primary Studies

	Level		**Reason**				**Technique**		**Time**		**Adaptation Mechanism**	
Paper	Server	Client	Context	Technical Resources	FL Client feedback	User Action	Compositional	Parameter	Proactive	Reactive	Approach	Managed Resources
[71]	×	✓	×	✓	×	×	Addition/Removal component	×	×	✓	Internal/External	(Physical) Gateway
[72]	✓	×	×	×	✓	×	×	Federated	✓	×	External	(Physical) FL server
[73]	✓	×	×	×	×	✓	Change component	×	✓	×	External	(Physical) Small device
[85]	×	✓	✓	×	×	×	Change component	Model	×	✓	External	(Physical) Gateway
[86]	✓	×	✓	×	×	×	Change component interaction	Federated	×	✓	Internal/External	(Physical) FL server
[87]	×	✓	×	×	✓	×	Change component	Federated	×	✓	External	(Physical) Small device
[88]	×	✓	✓	×	×	×	×	Model	×	✓	Internal	(Physical) FL server
[22]	×	✓	✓	×	×	×	×	Federated	×	✓	External	(Physical) FL server
[76]	✓	✓	✓	×	×	×	×	Model	×	✓	Internal/External	(Physical) FL server
[18]	×	✓	✓	×	×	×	Change component interaction	×	×	✓	Internal	(Physical) FL server
[89]	×	✓	×	×	✓	×	×	Federated	×	✓	External	(Physical) FL server
[90]	×	✓	✓	×	×	×	×	Model	×	✓	External	(Physical) FL server
[69]	✓	×	✓	×	×	×	Change component	×	×	✓	External	(Physical) FL server
[74]	×	✓	×	×	✓	×	Change component interaction	×	×	✓	Internal/External	(Physical) Gateway
[75]	✓	×	×	×	×	✓	Change component interaction	×	×	✓	External	(Virtual) Resource capability
[91]	✓	×	×	×	✓	×	×	Federated	×	✓	External	(Physical) FL Server
[92]	✓	×	✓	×	×	×	Change component interaction	×	×	✓	External	(Physical) FL Server

Adaptation logic The primary objective of adaptation logic is to explain how SAS can adapt to a given context. On the basis of the nature of IoFT, internal and external adaptations can occur on the FL server and client because they are not tightly coupled systems. Therefore, internal approaches can integrate the application and adaptation logic. This approach changes the internal configuration of the IoFT. For example, an FL server hosted on a cloud internally configures its virtual resources to provision autoscaling infrastructure based on a specific threshold [93]. Additionally, an FL server may change the ML model during runtime due to a decrease in overall performance. Similarly, an FL client may change the local ML model hyperparameters based on concept drift [94]. The internal approach has two main drawbacks: (i) the cost of testing and maintaining the system and (ii) the knowledge required to perform internal adaptation is only sometimes available [8] [7]. Therefore, external approaches can overcome these limitations by separating the adaptation logic and managed resources and connecting them via different interfaces. Thus, adaptation logic uses an interface to acquire knowledge from the context and interact with managed resources [12]. The modularity of the external approach makes it more suitable for IoFT. A classic example of an external approach is an FL server that uses the TCP/IP communication protocol to obtain the local model parameters of the FL client and optimize the global FL model according to the client's feedback. In the previous example, the communication protocol used to establish communication is an interface that allows the adaptation logic in FL servers to acquire knowledge from the context, which in this case is the FL client. Additionally, the FL client, in this context, uses the gateway as a physically managed resource to respond to the FL server. The external approach can also be implemented for FL clients. The FL client adaptation logic collects knowledge from the context via sensors and enables adaptation via actuators. For example, in federated smart homes with "anomaly detection" scenarios, the adaptation logic obtains humidity readings from the sensor, which provides the adaptation logic with contextual information to decide whether to open or close windows using an actuator. In summary, managed resource and adaptation logic are the two main pillars of SAS development, including IoFT. In Section 4.3, we explain the adaptation feedback loop for self-adaptive IoFT in detail.

4.3 Feedback Control Loop for Self-adaptive IoFT

The characteristics of self-adaptation systems make it crucial to understand how self-adaptive IoFT is used in MAPE-K loops. Furthermore, the feedback control loop mechanism should be able to manage the increasing complexity of IoFT systems. Therefore, the MAPE-K loop should contain an adaptation logic to manage different internal and external resources to comply with these standards. Unlike the traditional design of standalone and distributed systems, the IoFT system design must contain two modules, i.e., the FL server and client, to allow the IoFT system to function, as discussed in 2.2.1. To differentiate FL from a traditional client/server system, the minimum number of FL clients is two, each serving as an independent module in the FL system. Therefore, the FL client is equivalent to a standalone IoT system. The adaptation logic could be in the system gateway, and the managed resources could be virtual or physical within the IoT environment, as discussed in 4.2.5. In contrast, an FL server has different characteristics and managed resources to realize self-adaptation. To explain and analyze self-adaptation in the IoFT context, we used architecture-based self-adaptation [15] using the MAPE-K framework [12] to explain several aspects of the adaptation mechanism. First, we analyzed each IoFT level in section 4.2.3, including the adaptation logic and managed resources for both the FL server and client. Finally, we conclude by a generalized interpretation of MAPE-K for the entire FL system. Figure 6 shows our conceptual architecture of the proposed self-adaptive IoFT.

4.3.1 FL Client as a Managed Resource

On the FL client side, we categorize the devices into two types: nonadaptive and adaptive IoT client. The nonadaptive IoT client includes all small and large devices, such as smartphones, tablets, PCs, microcontrollers, microprocessors, and other embedded systems. An important criterion for nonadaptive devices is their ability to participate in FL collaboration. Sensors and devices unable to perform computations will be excluded unless they use an adapter to extend their capabilities, such as humidity and temperature sensors connected to a microcontroller to join in FL rounds. Another distinguishing feature of nonadaptive IoT clients is that they only train local models and share updated versions. They did not adapt to the context or modify the model parameters based on factors such as concept drift. In contrast, adaptive IoT clients are devices that can join FL collaboration and adapt to the given context. These FL clients use one of the self-adaptation approaches mentioned in 2.1. The global model is received from the FL server, and the local model is then trained using the local data. Asynchronously, they adapt the training behavior based on the sensing layer, which reflects the local model training. At the end of each round, all nonadaptive and adaptive IoT clients upload their trained local models to the FL server to join another round of training.

4.3.2 Middleware

The middleware is responsible for communication between the FL client and server for (1) uploading or downloading the FL model, (2) registering or removing an FL client, and (3) modifying an FL client's parameters. It uses communication

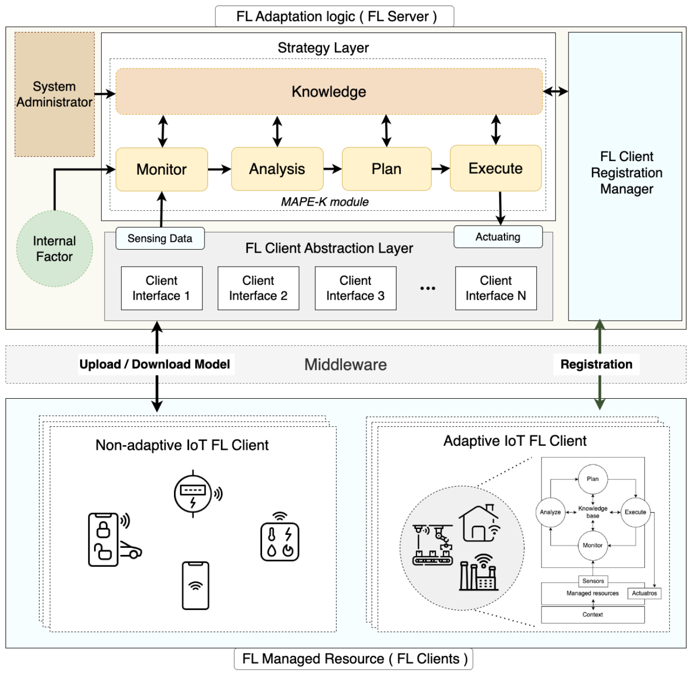


Figure 6: Conceptual architecture overview for self-adaptive IoFT

protocols to maintain the orchestration of FL, such as hypertext transfer protocol secure (HTTPS) and remote procedure call (RPC). HTTPS is based on TCP/IP, which leads to more reliable and stable communication between the FL server and clients. However, RPC uses TCP for reliability and order and UDP for low latency and overhead, depending on the application's requirements or the system's configuration. For example, TCP may be used when the amount of data to be sent cannot fit into a single UDP datagram. Otherwise, RPC will initially use UDP for small data.

4.3.3 FL Server as Adaptation Logic

The workflow of the self-adaptive feedback loop in the FL server has different characteristics and features than that in the federated client. The overall objective of the self-adaptive FL server is to maintain system quality by managing the orchestration of FL. Therefore, the FL server is responsible for the adaptation logic and can be installed on physical devices such as computers, microcontrollers, or virtual devices like containers and VMs running in the cloud. The FL server consists of several layers of independent and dependent components that form the FL adaptation logic.

The *FL client abstraction layer* is the interface between the FL server and clients, where each FL client is connected to a client interface to facilitate communication. The FL client proxy can be changed on the basis of the communication protocol used by the FL system. The data received from the FL clients via the client interface are used as contextual information for the MAPE-K module within the *strategy layer*. The *FL client registration manager* component provides a client registration or elimination mechanism that allows FL clients to join FL cooperation. The data are available in the knowledge base of the MAPE-K module and can be used later during the MAPE-K cycle. The *system administration* component allows the system administrator to interact with the FL system and modify FL parameters in the knowledge base. The *strategy layer* is the core layer containing the MAPE-K module, which helps the FL server implement model aggregations and maintain system performance. The MAPE-K cycle starts at the monitoring phase to collect data from the context, as discussed in 4.1.1. From the FL server's perspective, the context can be either physical resources (adaptive or nonadaptive IoT clients) or internal factors as virtual resources (CPU and RAM usage) as discussed in 4.2.5. The contextual information is collected in two modes, which differ depending on the data source. Data generated from virtual resources updated by the system administrator in the knowledge base are monitored in real-time. Conversely, the data received from the *FL client abstraction layer* are monitored after each FL round.

All received data were preprocessed in the *monitor phase* using the assumptions and system rules predefined in the knowledge base component. This preprocessing includes extracting data from various sources such as network instrumentation, environmental sensors, and application requirements, selecting a model (such as an NN model, ensemble ML model, or reinforcement learning [RL] model), and identifying FL clients (by participant information and type). After preprocessing, the sensing data are passed to the *analysis phase* for data diagnosis and model aggregation.

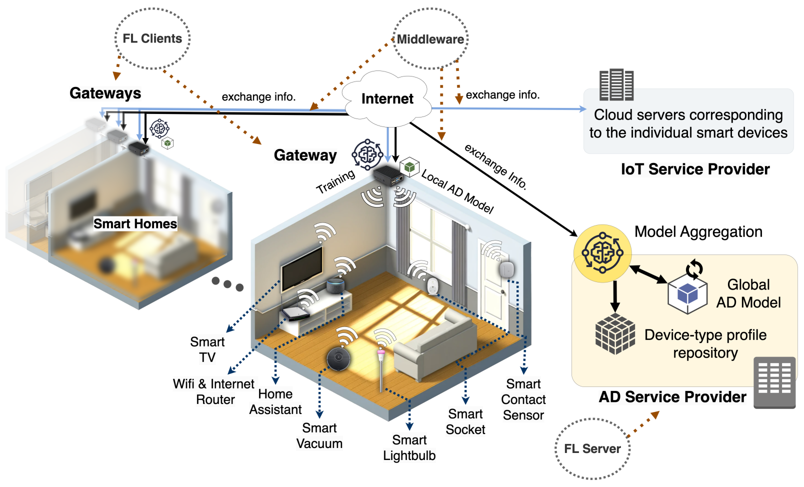


Figure 7: IoFT architecture for AD in smart homes

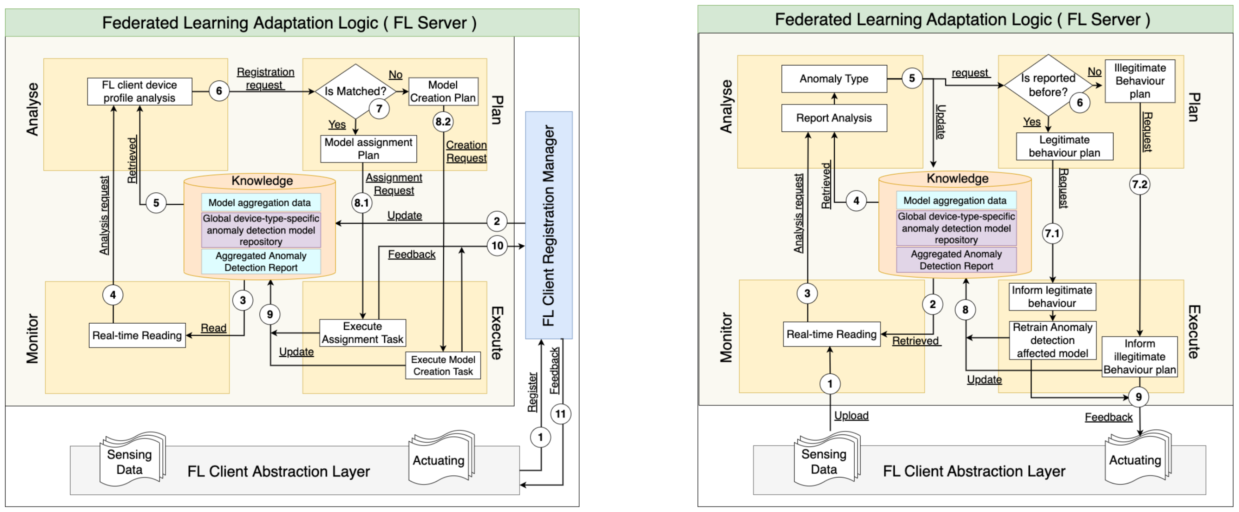
The sensing data received during the monitoring phase are used to analyze the quality of the FL client contributions. Depending on the system requirements, FL client configuration type, and targeted application, various reasoning algorithms can be used in the analysis phase. The required data for reasoning, training, and model aggregation tasks can be obtained from a knowledge base. The analysis task can also be extended to include multiple tasks simultaneously, such as fault detection and classification, as shown in [71]. The results of the data analysis are stored in a shared knowledge base, which makes them available across all MAKE-K framework components. If adaptation is required, an adaptation request is sent to the *plan phase*. In the planning phase, the system creates a workflow of adaptation actions required to maintain the system's performance. This workflow may include removing weak FL clients' contributions, reducing the weight of their contributions, changing the aggregation model approach, and changing the FL global configuration (such as the number of FL clients or the training rounds). The *execution phase* then conducts the workflows produced in the planning phase through the actuators of the managed resources.

Finally, the *FL client registration manager* component is the main starting point for any FL client interested in joining the collaboration. This component is responsible for initializing and profiling newly added FL clients and reporting them to the knowledge base within the *strategy layer*. Additionally, it provides the initial FL model and other FL configuration settings to the FL client. During the joining process, a new FL client has the opportunity to provide additional information regarding their context, which may help the FL to be aware of the context of the FL client. By introducing this component, the system designer will have extra flexibility to manage the initial data exchange between the FL client and server before allowing the new FL client to join the training round.

4.4 Self-adaptive IoFT for AD in Smart Home Use Cases

This section presents the conceptual architecture for self-adaptive IoFT for AD classification. AD involves finding unexpected items or events that deviate from the norm. It has received much attention from the research community. Nguyen et al. [71] propose DÏoT, a self-learning FL-based AD system for IoT. The definition of self-adaptation in their study is presented in Table 6. The system proposed by Nguyen et al. [71] successfully used FL to create ADs that could identify various IoT devices and build detection profiles based on device-type for small and home offices. Their system is capable of functioning without human intervention or labeled data. Therefore, it is essential to understand how IoFT for AD works in typical smart places. Figure 7 shows the architecture of using self-adaptive IoFT for AD in smart homes. This use case has two main components: (1) a security *gateway* as an FL client and (2) an *AD service provider* as an FL server. Security *gateway* has access to the internet and is responsible for running local AD to identify any infected IoT device. Additionally, it detects the type of device for any newly connected IoT device to the network to create a device-type profile. As part of FL participation, security *gateway* will train the received global model on local data and shares the model weight and the updated *device-type profile* at the end of each round. The AD service provider periodically supports the security *gateway*. Each device-type has an AD model stored in a model repository within the *AD service provider*. The security *gateways* within this proposed system are FL clients, whereas the IoT service provider is an FL server.

The proposed architecture includes security *gateways* that function as *adaptive IoT clients*. Their dedicated MAPE-K module allows them to self-learn the type of IoT device based on their communication pattern with the


Federated Learning Adaptation Logic (FL Server)
Analyse
Plan
Monitor
Execute
FL client device profile analysis
Registration request
Is Matched?
No
Yes
Model Creation Plan
Model assignment Plan
Creation Request
Assignment Request
Knowledge
Model aggregation data
Global device-type-specific anomaly detection model repository
Aggregated Anomaly Detection Report
Analysis request
Retrieved
Update
Feedback
Read
Real-time Reading
Execute Assignment Task
Execute Model Creation Task
FL Client Registration Manager
Register
Sensing Data
FL Client Abstraction Layer
Actuating
Anomaly Type
Report Analysis
request
Is reported before?
Illegitimate Behaviour plan
Legitimate behaviour plan
Request
Retrieved
Inform legitimate behaviour
Retrain Anomaly detection affected model
Inform illegitimate Behaviour plan
Upload

(a) automatic client enrollment (b) IoT device firmware update spotting

Figure 8: Two case studies of MAKE-K-based self-adaptive IoFT

security *gateways*. This self-learning AD is on the client side of the IoFT paradigm. However, the *AD service provider*, which acts as an FL server, can also automatically identify one of the challenges associated with the AD model based on device-type behavior. This challenge involves the rise of false alarms due to regular device firmware updates provided by one of *IoT service providers* in the market, such as Amazon, Samsung, and Philips. This behavior can affect the overall AD performance as it aggregates into a global model, which has a similar effect to that of a model poisoning attack [95]. Nguyen et al. [71] highlight this issue and propose that FL servers should track alarms that increase across FL clients to determine the legitimacy of the alarm. We incorporate this in our use case; therefore, the *AD service provider* has three tasks: (1) aggregating the FL client's module, (2) determining if a new *device-type profile* does not match any existing records in the *device-type profile* repository, and (3) dealing with IoT firmware updates that arise from FL clients with similar *device-type profiles*. The following two studies demonstrate *FL adaptation logic* interpretation within the proposed conceptual architecture of self-adaptive IoFT, explain the automatic enrollment of FL clients, and address IoT firmware updates for self-adaptive IoFT for AD in smart homes.

FL client automatic enrollment Figure 8a illustrates the complete cycle of *FL adaptation logic* for enrolling FL clients with existing or new device-type profiles.

- Monitor component: As the *FL registration manager* updates the *knowledge base* with the new FL client profile, the *monitor* component will be aware of any changes to the knowledge base. Therefore, the *monitor* component sends the analysis request to the *analysis component*.
- Analysis component: The new client registration request received from the *monitor* component will be analyzed during this phase. The *analysis* component retrieved the device-type-specific AD models from a repository in the *knowledge base*. The received profile is actively analyzed to match the new FL client to the existing AD based on the FL client's device-type profile. The *analysis* component sends a registration request to the *plan* component.
- Plan component: The *plan* component provides an action plan to achieve a specific goal. The primary task in this workflow is to register a new FL client is the primary task in this workflow. On the basis of the request received from the *analysis* component, the planning process decides whether the new FL client will obtain a previously trained model or start a new device-type-specific AD model. If the reported results match, the *plan* component will send a model assignment request to the *execute* component. Contrarily, if there is no match, the *plan* component will send a new FL client request instead.
- Execute component: In the *execution* component, two actions can be performed: (1) creating a new device-type-specific AD model and (2) assigning an existing model to the new FL client. The *execute* component will create a AD model if a new FL client request is received. The newly created model will also be uploaded to the shared *knowledge base* for future training. If a model assignment request has been received, the *execute* component will obtain the model from the *knowledge base* to be assigned to the FL client. The assigned or newly created model is reported to the *FL registration manager* component to return the model to the FL client.

IoT Device Firmware Update Spotting Figure 8b presents the complete cycle of the *FL adaptation logic* for updating the IoT device firmware. The process encompasses four phases.

- Monitor component: At the end of each training round, FL clients submit a report describing any anomalies detected in the updated FL model. These reports are forwarded to the *analysis* component for an in-depth examination.
- Analysis component: Upon receiving the reports, the *analysis* component first detects anomalies by retrieving anomaly information from the knowledge base to understand the context and potential implications of the reported issues. Following detection, the *analysis* component categorizes the anomaly by comparing it with a previously detected anomaly and reporting it to the *knowledge base*. The findings are reported to the *knowledge base* along with action requests to the *plan* component.
- Plan component: This component determines if the identified anomaly has been previously encountered. On the basis of this assessment, a response plan distinguishes between legitimate and illegitimate behaviors.
- Execute component: Two responses are initiated for legitimate anomalies: canceling false alarms triggered by firmware updates and retiring the specific AD model for the device-type. The *execute* component also includes updating the *knowledge base* to facilitate future model retraining. Conversely, for illegitimate anomalies, a verification process is initiated. The outcome of this stage is forwarded to the FL client via an actuator.

4.5 Motivation for Self-adaptive IoFT

This section examines the motivation behind previous studies on self-adaptive IoFT. A significant portion of these studies explicitly articulated motivation as a benchmark for evaluating the effectiveness of their proposals. Nonetheless, a few studies have not overtly acknowledged the use of self-adaptive IoFT. Therefore, we reviewed each study and extracted the inherent “self-” feature to bridge the connection between the intended purpose and motivation. We list the primary motivations that motivated researchers to explore the domain of self-adaptive IoFT.

4.5.1 Improve FL Performance

One of the core components of FL is predictive modeling. Unlike the conventional method of training and evaluating an ML model in a centralized manner, an FL model is trained and evaluated in a distributed environment, which can present many difficulties, as discussed in 2.2.1. Some studies in the literature has been conducted to improve the IoFT performance using a self-adaptation approach, as shown in Table. 9.

4.5.2 Data Quality

One of the challenges of the IoFT is data quality, which is used to train the global model. For example, IoFT based on stochastic gradient descent (SGD) is highly sensitive to the data used to train the model. Therefore, an independent and identically distributed (IID) data sample is required to ensure that the stochastic gradient is an unbiased estimate of the entire gradient. The non-IID data samples must be addressed to realize better performance [76]. To address this issue, several researchers have developed self-adaptive IoFT to minimize the effects of non-IIDs.

4.5.3 System Complexity and Resource Consumption

Because of the complexity of implementing an IoFT system in a real-world scenario, many researchers have attempted to facilitate the deployment and reduce the resource consumption of FL. This includes reformatting the structure of the IoFT system [73] or automatically altering training behavior [92]. A list of all primary studies that contributed to this motivation is presented in Table 9.

4.5.4 IoT Heterogeneity

Another motivation discussed in the literature is dealing with FL client heterogeneity. In real-world applications, IoT devices are diverse and have heterogeneous computing resources. Also, wireless networks are unstable, which makes it difficult to guarantee consistent connections as communication conditions change. Therefore, to implement FL training processes on heterogeneous FL client devices, researchers have advocated different self-adaptive approaches, ranging from semisupervised learning [71] to self-learning FL [89] [92].

4.6 Implementation of Self-adaptive IoFT

Engineering self-adaptive IoFT aims to encode “self–” properties within the IoFT system design to provide reactive or proactive action for managing the system state, knowledge, and execution environment. This section details the implementation of how self-adaptation techniques in IoFT systems.

Table 9: Motivation of self-adaptive IoFT systems

Motivation	Primary Studies
Improve FL performance	[71] [72] [85] [86] [88] [22] [76] [18] [90] [69] [74] [75] [91] [92]
Data Quality	[76] [18] [92] [86]
System Complexity and Resource Consumption	[73] [86] [87] [89] [74] [75] [92]
IoT Heterogeneity	[71] [72] [85] [87] [88] [22] [89] [69] [74] [75] [91] [92]

Table 10: Implementation of self-adaptive IoFT systems

Implementation	Primary Studies
Context-based approach	[71] [91]
ML-based approach	[85] [86] [88] [22] [76] [18] [90] [69] [75] [92]
Nature-based approach	[73] [87] [74]
Agent-based approach	[72] [89]

4.6.1 Context-based Approach

As discussed in section 4.1.1, the primary responsibility of context-aware systems is to assess the context and respond to dynamic environment changes to accomplish a specific objective based on relevant data. Nguyen et al. [71] use a semilearning approach to improve the overall AD classifier in an IoFT environment. They use a device-type identification approach to identify the device-type and extract feature representations of normality. As the authors claim, they have achieved no false alarms in a real-world smart home environment. Similarly, Zhaohang et al. [91] improve the quality of the FL performance by introducing an adaptive asynchronous FL for participants. They solve the problem of slower devices that decrease convergence speed and worsen global model performance.

4.6.2 ML-based Approach

Learning and self-adaptation are closely related in IoFT systems. A self-adaptive IoFT system continuously adjusts its structure, settings, or algorithms to increase its effectiveness, as discussed in 4.2.4. Therefore, various learning techniques have been used in the literature to achieve different system goals. First, improving the ML performance in the IoFT was a motivation for using the self-adaptation technique. For example, Li et al. [22] take advantage of information about a device's historical training tasks to constrain future workflow and combine it with active learning to select participants adaptively. They also integrate spatiotemporal information using a self-attention approach to mix local and global models based on differences between local, global, and personalized models [90]. Zhu et al. [69] propose a mechanism to improve the uploaded weight of parameters to the FL server. They also achieved faster convergence with higher accuracy by improving the sparsity of the global model by continuously updating the online optimization function in the proposed algorithm. In contrast, Baresi et al. [75] explore the self-adaptive IoFT system by optimizing client resources at runtime considering network overhead and model accuracy. Second, there are studies to improve the quality of data collected from FL client. For example, Duan et al. [86] use the Kullback–Leibler divergence to solve unbalanced data and improve FL performance. Similarly, Wang et al. [18] introduced self-paced learning to retain high-confidence samples and eliminate high-noise samples. Most FL algorithms are severely affected by data distribution; thus, Xu et al. [76] proposed a self-attention approach to set up a communication strategy for FL effectively. Their approach applies parameter optimization for server-to-client and client-to-server to overcome the problem of non-IID in FL settings. Finally, some work was conducted to address the unlabeled data collected from the FL client. Saeed et al. [85] leverage the scalogram signal correspondence learning via wavelet transform to self-learn valuable representations from unlabeled sensor inputs. He et al. [88] propose a self-supervised and personalized FL framework that employs algorithms to train global and personalized models. Moreover, Chen et al. [92] propose a federated graph learning framework to optimize a global self-supervision model to create global pseudo-label discovery and graph construction that are shared with a federated client to label data.

4.6.3 Nature-based Approach

A nature-inspired approach optimizes a given system to achieve specific goals. The main idea is that each system component has limited information and acts according to the tasks assigned by the system administrator. The overall system behavior can be predicted on the basis of the behavior of each individual participating in the population. Reducing training time in ML is a significant area of interest in nature-inspired computing. Khan et al. [73] propose a novel FL scheme that uses a developed optimization problem to minimize the global FL training time. They achieve this with the help of cluster formalization, joint device frequency selection, and resource allocation. Another study was conducted by Lim et al. [87] to select an FL server and allocate resources based on evolutionary games. Their approach provides two levels. The lower level uses an evolutionary game to represent workers' edge association strategies. With the edge server's bandwidth allocation control method, the upper level uses a Stackelberg differential game in which the model owner selects the optimal reward structure. Franco et al. [74] provide a self-adaptive architecture for IoFT in industrial automation systems. Their method considers participating parties at various industrial ecosystem levels. Each factory internally trains the model in a self-adaptive manner and delivers it to the centralized cloud server for global aggregation to fulfill the goals of global model optimization and reduction of communication cycles. Furthermore, to overcome the multi-assignment optimization problem, the researchers split the dataset into subsets equivalent to the number of participants. Each device selects an appropriate subset during each local iteration to optimize the model.

4.6.4 Agent-based Approach

The agent-based approach involves agents that share similar goals. These agents coordinate and communicate to facilitate efficient functioning in decentralized systems. This approach can be helpful in the IoT environment because different IoT devices can be considered different agents. Pang et al. [72] use the agent-based approach in IoFT to find cooperation plans based on RL. They also adjust the system weight based on user feedback. Similarly, Tam et al. [89] proposed implementing a self-learning agent that engages with a network function virtualization (NFV) orchestrator and a software-defined networking (SDN) architecture by incorporating a deep q-learning algorithm.

4.7 Evaluation of self-adaptive IoFT

In this section, we examine the evaluation methodologies used in the literature for self-adaptive IoFT and the testbeds involved.

4.7.1 IoFT Testbeds

The proposed IoFT system can be implemented and evaluated using different approaches. Building a case study technique allows multifaceted examinations of complex topics in real-life contexts. In the scientific world, the case study method is highly valued. For instance, Khan et al. [73] use smart tourism as a case study that involved tourists using smart devices to visit historical sites. They proposed an approach to automatically formalize FL device clusters and resource allocation in IoFT environments. Another approach to building a testbed is to implement an IoFT system at a physical location. This approach explores a system's behavior in a real-world setting, which increases its credibility. Moreover, we can build a local laboratory network to collect datasets using VMs and physical machines. Nguyen et al. [71] detects Mirai malware in small and home offices. They generated 33 profiles for IoT devices in a local laboratory to evaluate the proposed framework for self-learning and AD. Additionally, we can use a simulation approach that mimics the actual behavior of an existing system. For example, Tam et al. [89] implement a self-adaptive resource optimization approach for FL client by simulating an IoFT environment using Mininet, which is a virtual network emulator. Another widely adopted approach in the research community is using publicly available datasets. In the literature, various datasets have been used, such as MNIST [75] [91] [76] [86], CIFAR10 [91], CINIC-10 [86] [88] [76], BAL1 [86], FEMNIST [22], Yelp [90], Tweets [90], and Cora [92].

4.7.2 IoFT Model Evaluation Criteria

The evaluation criteria were similar to those of the conventional ML paradigms. Most primary studies have used performance metrics to evaluate self-adaptive IoFT. These metrics include FL model accuracy [72] [86] [88] [22] [76] [18] [89] [90] [69] [74] [75] [91] [92], false positives [71], true positives [71], F-score [85] [18], Kappa [85], training loss [88] [22] [76] [89] [75], AUC [18], recall [18], precision [18], negative log-likelihood [90], and average percentage ranking [90]. To evaluate system performance, resource consumption, and network overhead, researchers have used different metrics, such as bandwidth [87], packet drop and delivery ratio, and delay [89].

4.8 Application of Self-adaptive IoFT

This section presents the recognized application domains that may benefit from using the concept of self-adaptive IoFT in the literature.

4.8.1 Health Care

Health care faces significant challenges, including patient privacy concerns, data availability issues, and system complexities. For instance, IoT devices collecting patient metrics often encounter restrictions on data sharing across institutions for further analysis. However, limitations, such as obtaining patient information, restrict ML-based healthcare applications from scaling effectively. Consequently, self-adaptive IoFT offers a solution by enabling self-adaptive collaborative model training across institutions without sharing patient data to preserve privacy and governance

[96]. A self-adaptive IoFT involves handling data collection and energy efficiency-related issues and developing an unbiased model for better diagnosis and treatment. For example, Wang et al. [18] demonstrate self-adaptation in patient data to diagnose diseases while preserving patient privacy.

4.8.2 IoFT for Dynamic Context

Self-adaptive IoFT can be used in unstable environments that allow dynamic replacement or adjustment of the IoFT compositional components (see 4.2.4) according to the context (see 4.1.1), which the knowledge acquisition of the target environment can drive. Self-adaptive IoFT can maintain system performance and availability by applying the aforementioned techniques. For example, Khan et al. [73] present a self-organizing FL over a wireless network in smart tourism to substitute an unavailable FL server with another candidate, and Lim et al. [87] propose FL client association strategies and resource allocation to reduce communication overhead for large-scale implementation.

4.8.3 Industrial IoT (IIoT)

Industrial systems use distributed control systems, supervisory control and data acquisition, and programable logic controllers to provide real-time collaboration between sensors, actuators, and controllers. These systems must communicate with each other to maintain the quality of service (QoS). Additionally, real-time feedback is important in some applications, such as cyber-physical systems. Consequently, most of these systems run inference ML models on edge . Franco et al. [74] show an example derived from the IIoT domain, where they proposed a self-adaptive IoFT architecture for industrial automation systems.

4.8.4 Smart-*

Most smart-* applications, such as smart places, cities, and homes, use heterogeneous IoT devices to satisfy user requirements. These requirements may be related to user comfort and privacy or the prevention of cyberattacks on smart applications. For example, Nguyen et al. [71] present a self-learning AD model for IoFT to detect Mirai malware in smart offices.

4.9 Challenges and Future Direction

Using self-adaptive IoFT brings new features to current FL and IoT systems. The goal of this utilization is to provide extra resiliency to IoFTs. However, some challenges need to be addressed to achieve the full benefits of embodying this new design paradigm.

4.9.1 Challenges in Designing Self-adaptive IoFT

One of the greatest challenges is identifying the requirements of the IoFT system that must be maintained and how to build a self-adaptive IoFT that fulfills this adaptation requirement. In Section 4.2, we explain the main properties of the self-adaptive IoFT design. As a part of the *technique* used, parameter adaptation in IoFT can be challenging. An engineer who designs self-adaptive IoFT needs to clearly state which parameters need to be optimized automatically (FL global parameter, model initial parameter, and model hyperparameter). An optimization technique based on a nature-based approach can be used to modify these parameters, as discussed in Section 4.6.3. However, attempting to change all parameters in the IoFT paradigm at the same time is a resource-intensive task. Moreover, it can be clearly observed that we did not find any work to perform self-adaptation *proactively*. Setting a strict adaptation policy to drive adaptation toward a specific direction makes *reactive* adaptation more appealing for system designers because it triggers the adaptation process when these policies are violated. More work on *proactive* adaptation is required, especially for critical applications (health care, finance, and IIoT), where we must start the adaptation process before the violation occurs. Qi et al. [97] follow this direction by introducing proactive handover using FL to maintain QoS for mobile users in vehicular networks.

4.9.2 Challenges in FL Nature

Self-adaptive IoFT is also prone to challenges affecting FL systems. The heterogeneity of data, statistics, and devices across different FL clients increases the complexity of defining the adaptation policy. Because of data privacy constraints, sharing information about the context is not usually possible. In Section **??**, we demonstrate the benefit of sharing the device-type profile across different FL clients to identify false alarms arising from firmware updates on IoT device. Another challenge is the limited availability of the FL client. The number of FL clients can vary depending on the type of FL architecture used. For example, the number of FL clients in cross-device FL can be large, and many are only sometimes available to participate in training. Khan et al. [73] introduced self-organizing FL over wireless networks that autonomously formalizes device clusters, joins devices, and allocates resources to overcome the issue of FL client availability. These challenges must be considered when designing a self-adaptive IoFT. A potential approach is to design a new policy, such as a client selection and aggregation approach that considers the challenges associated with the traditional FL application.

4.9.3 Challenges in Adaptation Logic

The MAPE-K framework is the main engine that drives the IoFT toward different states. To use them effectively, we need to highlight the difficulty of each stage of the framework. For instance, which part of the IoFT system should be *monitored*? What is the state and behavior of IoFT that should be *analyzed* thoroughly to identify the system's status? What is the best adaptation *plan* for the underlying system to make effective decisions? What is the correct execution approach that assures the adaptation is performed correctly? Answering these questions required a comprehensive understanding of the specific IoFT system being used to monitor, analyze, plan, and execute adaptations effectively. Also, it requires in-depth knowledge of the system's components, functionalities, and potential challenges that may arise during each framework stage. Jahan et al. [98] present a security-focused feedback control loop that interacts with the MAPE-K framework to dynamically manage runtime adaptation to changes in functional and security conditions. Their approach provides two control loops that interact with each other to reduce the effort undertaken at the design stage for building a reliable MAPE-K framework. While some attempts have been made to address the challenges of designing the MAPE-K framework in other domains, this topic requires further attention when applied to a self-adaptive IoFT system.

5 Conclusion

Self-adaptive IoFT involves a combination of SAS, FL, and IoT. In this study, we thoroughly summarize the current initiatives to use self-adaptive IoFT. We specifically examine many computer fields that contributed to this effort, including FL, SAS, feedback loop controllers, and IoT. Furthermore, we provide a multidimensional taxonomy that highlights the key features of a self-adaptive IoFT system. Additionally, we present and apply a conceptual architecture for self-adaptive IoFT. We introduce a use case of applying self-adaptive IoFT for AD in smart homes, mainly how automatic FL clients join the collaboration and how we can spot the rise in false alarms from IoT device firmware update in a collective of smart homes. Moreover, we outline the motivation, implementation, evaluation, and real-world applications of the self-adaptive IoFT. Finally, we discuss the challenges and future research directions, which will be valuable for researchers and developers of this discipline.

References

[1] Sandro Nižetić, Petar Šolić, Diego López-de-Ipiña González-de-Artaza, and Luigi Patrono. Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future. *Journal of Cleaner Production*, 274:122877, nov 2020.

[2] Semiconductor Digest. Number of connected iot devices will surge to 125 billion by 2030, 2017.

[3] Kaiqiang Qi and Chenyang Yang. Popularity prediction with federated learning for proactive caching at wireless edge. In *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2020.

[4] Raed Kontar, Naichen Shi, Xubo Yue, Seokhyun Chung, Eunshin Byon, Mosharaf Chowdhury, Judy Jin, Wissam Kontar, Neda Masoud, Maher Noueihed, Chinedum E. Okwudire, Garvesh Raskutti, Romesh Saigal, Karandeep Singh, and Zhisheng Ye. The internet of federated things (ioft): A vision for the future and in-depth survey of data-driven approaches for federated learning, 2021.

[5] Petr Jan Horn. Autonomic computing: Ibm's perspective on the state of information technology. 2001.

[6] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, 40(3), aug 2008.

[7] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2), may 2009.

[8] Christian Krupitzer, Felix Maximilian Roth, Sebastian Vansyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17(PB):184–206, feb 2015.

[9] Abdessalam Elhabbash, Maria Salama, Rami Bahsoon, and Peter Tino. Self-awareness in Software Engineering. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 14(2), oct 2019.

[10] Sawsan Abdulrahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. A Survey on Federated Learning: The Journey From Centralized to Distributed On-Site Learning and Beyond. *IEEE Internet of Things Journal*, 8(7):5476–5497, apr 2021.

[11] Rima Al-Ali, Lubomír Bulej, Jan Kofroň, and Tomáš Bureš. A guide to design uncertainty-aware self-adaptive components in Cyber–Physical Systems. *Future Generation Computer Systems*, 128:466–489, mar 2022.

[12] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[13] David Garlan, S.-W. Cheng, A.-C. Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, oct 2004.

[14] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(2):223–259, dec 2006.

[15] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovc, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications*, 14(3):54–62, 1999.

[16] Muhammad Asad, Ahmed Moustafa, and Takayuki Ito. Federated learning versus classical machine learning: A convergence comparison, 2021.

[17] Chang Liu, Shaoyong Guo, Song Guo, Yong Yan, Xuesong Qiu, and Suxiang Zhang. Ltsm: Lightweight and trusted sharing mechanism of iot data in smart city. *IEEE Internet of Things Journal*, 9(7):5080–5093, 2022.

[18] Qingyong Wang and Yun Zhou. FedSPL: federated self-paced learning for privacy-preserving disease diagnosis. *Briefings in Bioinformatics*, 23(1), jan 2022.

[19] Yang, Liu, Zhang, Junxue, Chaiand Di, Wangand Leye, Guo, Kun, Chen, Kai, Yang, and Qiang. Practical and secure federated recommendation with personalized masks, 2021.

[20] Yunfan Ye, Shen Li, Fang Liu, Yonghao Tang, and Wanting Hu. Edgefed: Optimized federated learning based on edge computing. *IEEE Access*, 8:209191–209198, 2020.

[21] Su, Zhou, Wang, Yuntao, Luan, Tom H., Zhang, Ning, Li, Feng, Chen, Tao, Cao, and Hui. Secure and efficient federated learning for smart grid with edge-cloud collaboration. *IEEE Transactions on Industrial Informatics*, 18(2):1333–1344, 2022.

[22] Li Li, Moming Duan, Duo Liu, Yu Zhang, Ao Ren, Xianzhang Chen, Yujuan Tan, and Chengliang Wang. FedSAE: A Novel Self-Adaptive Federated Learning Framework in Heterogeneous Systems. In *2021 International Joint Conference on Neural Networks (IJCNN)*, volume 2021-July, pages 1–10. IEEE, jul 2021.

[23] Rajesh Kumar, Abdullah Aman Khan, Jay Kumar, Zakria, Noorbakhsh Amiri Golilarz, Simin Zhang, Yang Ting, Chengyu Zheng, and Wenyong Wang. Blockchain-federated-learning and deep learning models for covid-19 detection using ct imaging. *IEEE Sensors Journal*, 21(14):16301–16314, 2021.

[24] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. 2016.

[25] David Leroy, Alice Coucke, Thibaut Lavril, Thibault Gisselbrecht, and Joseph Dureau. Federated learning for keyword spotting. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6341–6345, 2019.

[26] Joel Stremmel and Arjun Singh. Pretraining federated text models for next word prediction, 2020.

[27] Rui Zhang, Bayu Distiawan Trisedy, Miao Li, Yong Jiang, and Jianzhong Qi. A benchmark and comprehensive survey on knowledge graph entity alignment via representation learning, 2021.

[28] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems*, 36(6):87–98, 2021.

[29] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[30] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. A Secure Federated Transfer Learning Framework. 2020.

[31] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions, 2018.

[32] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G.L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecní, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and Open Problems in Federated Learning. *Foundations and Trends in Machine Learning*, 14(1-2):1–210, dec 2019.

[33] Andrew Hard, Chloé M Kiddon, Daniel R Ramage, Françoise Simone Beaufays, Hubert Eichner, Kanishka Rao, Rajiv Mathews, Sean Augenstein, and Swaroop Ramaswamy. Federated learning for mobile keyboard prediction, 2019.

[34] Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. Protection against reconstruction and its applications in private federated learning, 2019.

[35] Shihong Huang and Pedro Miranda. Incorporating human intention into self-adaptive systems. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 571–574, 2015.

[36] Huihua Lu and Bojan Cukic. An adaptive approach with active learning in software fault prediction. In *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, PROMISE '12, page 79–88, New York, NY, USA, 2012. Association for Computing Machinery.

[37] Dhrgam AL Kafaf and Dae-Kyoo Kim. A web service-based approach for developing self-adaptive systems. *Computers & Electrical Engineering*, 63:260–276, 2017.

[38] Bogdan Solomon, Dan Ionescu, Marin Litoiu, and Mircea Mihaescu. A real-time adaptive control of autonomic computing environments. In *Proceedings of the 2007 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '07, page 124–136, USA, 2007. IBM Corp.

[39] Ivan Dario Paez Anaya, Viliam Simko, Johann Bourcier, Noël Plouzeau, and Jean-Marc Jézéquel. A prediction-driven adaptation approach for self-adaptive sensor networks. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2014, page 145–154, New York, NY, USA, 2014. Association for Computing Machinery.

[40] Xing Chen, Junxin Lin, Bing Lin, Tao Xiang, Ying Zhang, and Gang Huang. Self-learning and self-adaptive resource allocation for cloud-based software services. *Concurrency and Computation: Practice and Experience*, 31(23):e4463, 2019. e4463 CPE-17-0360.

[41] Nathalia Moraes do Nascimento and Carlos José Pereira de Lucena. Fiot: An agent-based framework for self-adaptive and self-organizing applications based on the internet of things. *Information Sciences*, 378:161–176, 2017.

[42] Ryan Heartfield, George Loukas, Anatolij Bezemskij, and Emmanouil Panaousis. Self-configurable cyber-physical intrusion detection for smart homes using reinforcement learning. *IEEE Transactions on Information Forensics and Security*, 16:1720–1735, 2021.

[43] Bihuan Chen, Xin Peng, Yijun Yu, Bashar Nuseibeh, and Wenyun Zhao. Self-adaptation through incremental generative model transformations at runtime. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 676–687, New York, NY, USA, 2014. Association for Computing Machinery.

[44] Matthias Hölzl and Thomas Gabor. Continuous collaboration: A case study on the development of an adaptive cyber-physical system. In *2015 IEEE/ACM 1st International Workshop on Software Engineering for Smart Cyber-Physical Systems*, pages 19–25, 2015.

[45] Arthur Rodrigues, Genaína Nunes Rodrigues, Alessia Knauss, Raian Ali, and Hugo Andrade. Enhancing context specifications for dependable adaptive systems: A data mining approach. *Information and Software Technology*, 112:115–131, 2019.

[46] Theresia Ratih Dewi Saputri and Seok-Won Lee. The application of machine learning in self-adaptive systems: A systematic literature review. *IEEE Access*, 8:205948–205967, 2020.

[47] Sin Kit Lo, Qinghua Lu, Chen Wang, Hye-Young Paik, and Liming Zhu. A systematic literature review on federated machine learning: From a software engineering perspective. *ACM Comput. Surv.*, 54(5), may 2021.

[48] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning, 2018.

[49] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. *Advances in Neural Information Processing Systems, Best Paper Award at Federate Learning Workshop*, 2020.

[50] Sebastian Caldas2018, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings, 2018.

[51] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2020.

[52] NVIDIA. Nvidia clara - nvidia developer, 2016. `https://developer.nvidia.com/clara`, Last accessed on 2023-08-08.

[53] Wang and Yanjun Ma anddianhai Yu andtian. PaddlePaddle: An Open-Source deep learning platform from industrial practice. *Frontiers of Data and Domputing*, 1(1):105–115, January 2019.

[54] G Anthony Reina, Alexey Gruzdev, Patrick Foley, Olga Perepelkina, Mansi Sharma, Igor Davidyuk, Ilya Trushkin, Maksim Radionov, Aleksandr Mokrov, Dmitry Agapov, Jason Martin, Brandon Edwards, Micah J. Sheller, Sarthak Pati, Prakash Narayana Moorthy, Shih han Wang, Prashant Shah, and Spyridon Bakas. Openfl: An open-source framework for federated learning, 2021.

[55] TensorFlow. Tensorflow federated, 2016. `https://www.tensorflow.org/federated`, Last accessed on 2023-08-08.

[56] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. 2019.

[57] Konstantin Burlachenko, Samuel Horváth, and Peter Richtárik. FL_PyTorch. In *Proceedings of the 2nd ACM International Workshop on Distributed Machine Learning*. ACM, dec 2021.

[58] Sachin Kumar, Prayag Tiwari, and Mikhail Zymbler. Internet of Things is a revolutionary approach for future technology enhancement: a review. *Journal of Big Data*, 6(1), dec 2019.

[59] Wondimagegn Mengist, Teshome Soromessa, and Gudina Legese. Method for conducting systematic literature review and meta-analysis for environmental science research. *MethodsX*, 7:100777, 2020.

[60] D Denyer and D Tranfield. The sage handbook of organizational research methods, 2009.

[61] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *J. Syst. Softw.*, 80(4):571–583, apr 2007.

[62] Martin Goller and Sven Tomforde. On the stability of (self-)adaptive behaviour in continuously changing environments: A quantification approach. *Array*, 11:100069, sep 2021.

[63] P J Brown. The stick-e document: a framework for creating context-aware applications. *Electronic Publishing 96*, 8, 1996.

[64] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In Hans-W. Gellersen, editor, *Handheld and Ubiquitous Computing*, pages 304–307, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[65] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE Communications Surveys & Tutorials*, 16(1):414–454, 2014.

[66] Smith Baker. The identification of the self. *Psychological Review*, 4:272–284, 5 1897.

[67] Goffman, Erving, , and University of Edinburgh. *The presentation of self in everyday life*. University of Edinburgh, Social Sciences Research Centre Edinburgh, 1956.

[68] Samuel Kounev, Peter Lewis, Kirstie L. Bellman, Nelly Bencomo, Javier Camara, Ada Diaconescu, Lukas Esterle, Kurt Geihs, Holger Giese, Sebastian Götz, Paola Inverardi, Jeffrey O. Kephart, and Andrea Zisman. The notion of self-aware computing. *Self-Aware Computing Systems*, pages 3–16, 2 2017.

[69] Meng yuan Zhu, Zhuo Chen, Ke fan Chen, Na Lv, and Yun Zhong. Attention-based federated incremental learning for traffic classification in the Internet of Things. *Computer Communications*, 185:168–175, mar 2022.

[70] L. Rodrigues, J. Guerreiro, and N. Correia. Resource design in federated sensor networks using RELOAD/CoAP overlay architectures. *Computer Communications*, 179:11–21, nov 2021.

[71] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N. Asokan, and Ahmad Reza Sadeghi. DIoT: A Federated Self-learning Anomaly Detection System for IoT. *Proceedings - International Conference on Distributed Computing Systems*, 2019-July:756–767, apr 2018.

[72] Junjie Pang, Yan Huang, Zhenzhen Xie, Qilong Han, and Zhipeng Cai. Realizing the Heterogeneity: A Self-Organized Federated Learning Framework for IoT. *IEEE Internet of Things Journal*, 8(5):3088–3098, mar 2021.

[73] Latif U. Khan, Madyan Alsenwi, Zhu Han, and Choong Seon Hong. Self Organizing Federated Learning Over Wireless Networks: A Socially Aware Clustering Approach. *International Conference on Information Networking*, 2020-January:453–458, jan 2020.

[74] Nicola Franco, Hoai My Van, Marc Dreiser, and Gereon Weiss. Towards a Self-Adaptive Architecture for Federated Learning of Industrial Automation Systems. *Proceedings - 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2021*, pages 210–216, may 2021.

[75] Luciano Baresi, Giovanni Quattrocchi, and Nicholas Rasi. Federated Machine Learning as a Self-Adaptive Problem. *Proceedings - 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2021*, pages 41–47, may 2021.

[76] HengJie Yawen Xu, Xiaojun Li, Zeyu Yang, Yawen Xu, and Hengjie Song. Robust communication strategy for federated learning by incorporating self-attention. *https://doi.org/10.1117/12.2581491*, 11584(10):336–341, nov 2020.

[77] Wei Yang Bryan Lim, Zehui Xiong, Chunyan Miao, Dusit Niyato, Qiang Yang, Cyril Leung, and H. Vincent Poor. Hierarchical Incentive Mechanism Design for Federated Machine Learning in Mobile Networks. *IEEE Internet of Things Journal*, 7(10):9575–9588, oct 2020.

[78] Jesper Andersson, Rogério De Lemos, Sam Malek, and Danny Weyns. Modeling dimensions of self-adaptive software systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5525 LNCS:27–47, 2009.

[79] Yuriy Brun, Ron Desmarais, Kurt Geihs, Marin Litoiu, Antonia Lopes, Mary Shaw, and Michael Smit. A Design Space for Self-Adaptive Systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7475 LNCS:33–50, 2013.

[80] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization, 2021.

[81] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2020.

[82] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates, 2021.

[83] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning, 2020.

[84] P.K. McKinley, S.M. Sadjadi, E.P. Kasten, and B.H.C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

[85] Aaqib Saeed, Flora D. Salim, Tanir Ozcelebi, and Johan Lukkien. Federated Self-Supervised Learning of Multi-Sensor Representations for Embedded Intelligence. *IEEE Internet of Things Journal*, 8(2):1030–1040, jul 2020.

[86] Moming Duan, Duo Liu, Xianzhang Chen, Yujuan Tan, Jinting Ren, Lei Qiao, and Liang Liang. Astraea: Self-balancing Federated Learning for Improving Classification Accuracy of Mobile Deep Learning Applications. *Proceedings - 2019 IEEE International Conference on Computer Design, ICCD 2019*, pages 246–254, jul 2019.

[87] Wei Yang Bryan Lim, Jer Shyuan Ng, Zehui Xiong, Dusit Niyato, Chunyan Miao, and Dong In Kim. Dynamic Edge Association and Resource Allocation in Self-Organizing Hierarchical Federated Learning Networks. *IEEE Journal on Selected Areas in Communications*, 39(12):3640–3653, dec 2021.

[88] Chaoyang He, Zhengyu Yang, Erum Mushtaq, Sunwoo Lee, Mahdi Soltanolkotabi, and Salman Avestimehr. SSFL: Tackling Label Deficiency in Federated Learning via Personalized Self-Supervision. oct 2021.

[89] Prohim Tam, Sa Math, Chaebeen Nam, and Seokhoon Kim. Adaptive Resource Optimized Edge Federated Learning in Real-Time Image Sensing Classifications. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:10929–10940, 2021.

[90] Anliang Li, Shuang Wang, Wenzhu Li, Shengnan Liu, and Siyuan Zhang. Predicting Human Mobility with Federated Learning. In *GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pages 441–444, New York, NY, USA, nov 2020. ACM.

[91] Wang Zhaohang, Xia Geming, Chen Jian, and Yu Chaodong. Adaptive Asynchronous Federated Learning for Edge Intelligence. *Proceedings - 2021 International Conference on Machine Learning and Intelligent Systems Engineering, MLISE 2021*, pages 285–289, 2021.

[92] Chuan Chen, Weibo Hu, Ziyue Xu, and Zibin Zheng. FedGL: Federated Graph Learning Framework with Global Self-Supervision. may 2021.

[93] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1:7–18, 5 2010.

[94] Firas Bayram, Bestoun S. Ahmed, and Andreas Kassler. From concept drift to model degradation: An overview on performance-aware drift detectors, 2022.

[95] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to Byzantine-Robust federated learning. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1605–1622. USENIX Association, August 2020.

[96] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletarì, Holger R. Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N. Galtier, Bennett A. Landman, Klaus Maier-Hein, Sébastien Ourselin, Micah Sheller, Ronald M. Summers, Andrew Trask, Daguang Xu, Maximilian Baust, and M. Jorge Cardoso. The future of digital health with federated learning. *npj Digital Medicine*, 3(1), September 2020.

[97] Kaiqiang Qi, Tingting Liu, and Chenyang Yang. Federated learning based proactive handover in millimeter-wave vehicular networks, 2021.

[98] Sharmin Jahan, Ian Riley, Charles Walter, Rose F. Gamble, Matt Pasco, Philip K. McKinley, and Betty H.C. Cheng. Mape-k/mape-sac: An interaction framework for adaptive systems with security assurance cases. *Future Generation Computer Systems*, 109, 2020.